Universidade Federal de Campina Grande

Centro de Ciências e Tecnologia

Departamento de Sistemas e Computação

Programa de Educação Tutorial

PET

Um Processo de Desenvolvimento de Software

Campina Grande, Fevereiro de 2007

Sumário

Índice de Figuras

Índice de Quadros

1. Introdução

A necessidade de se utilizar melhores práticas para desenvolvimento de software no meio acadêmico, que possibilitem maior sucesso na implementação de projetos oferecidos em algumas disciplinas, foi a principal motivação para a criação do easYProcess.

Sendo assim, o YP se apresenta como um processo simplificado, iterativo e incremental, apoiado em práticas do XP [1], RUP [2] e Agile Modeling [3]. Idealizado pela Professora Drª Francilene Procópio Garcia, uma das responsáveis pelo Grupo de Pesquisa e Desenvolvimento em Engenharia de Software do Departamento de Sistemas e Computação (DSC) [4] da Universidade Federal de Campina Grande (UFCG) [5], foi concebido no ambiente do grupo PET Computação [6], no ano de 2003.

A equipe do YP foi composta por 9 componentes, entre graduandos e alunos do PET. Na fase de desenvolvimento do processo, a equipe foi dividida em três grupos de estudos entre os processos XP, RUP e Agile Modeling, para que as práticas utilizadas por estes três processos fossem confrontadas no intuito de se extrair as melhores para a realidade do YP.

O YP vem sendo utilizado na disciplina de Laboratório de Engenharia de Software (LES) [7] desde o período 2003.1. Logo no início da sua implantação em LES, identificou-se a necessidade de melhorar o processo, na busca de torná-lo um processo robusto e capaz de suprir as necessidades das equipes de desenvolvimento que o utilizam. Nesse sentido, agregou-se ao fluxo do YP artefatos que possibilitam analisar aspectos relacionados à usabilidade do produto que está sendo desenvolvido.

Espera-se que o easYProcess possa auxiliar o gerenciamento do progresso no desenvolvimento de aplicações durante as disciplinas que assim necessitem, além de se tornar uma metodologia utilizada pelos alunos da graduação e que seja expandido para o uso em projetos de pequeno e médio porte em empresas.

2. Equipe de Desenvolvimento

Orientadora

Francilene Procópio Garcia............. .... garcia@dsc.ufcg.edu.br

Equipe de desenvolvimento:

Aliandro Higino Guedes Lima.......... ....aliandro@dsc.ufcg.edu.br

Danilo de Sousa Ferreira................ ....danilo@dsc.ufcg.edu.br

Fábio Luiz Leite Júnior.................. ....fabio@dsc.ufcg.edu.br

Giselle Regina Chaves da Rocha...... ....giselle@dsc.ufcg.edu.br

Gustavo Wagner Diniz Mendes........ ....gustavo@dsc.ufcg.edu.br

Renata França de Pontes.............. ....renata@dsc.ufcg.edu.br

Verlaynne Kelley da Hora Rocha..... ....verla@dsc.ufcg.edu.br

Vinicius Farias Dantas.................. ....vinicius@dsc.ufcg.edu.br

Yuska Paola Costa Aguiar............... ....yuska@dsc.ufcg.edu.br

3. Síntese do easYProcess
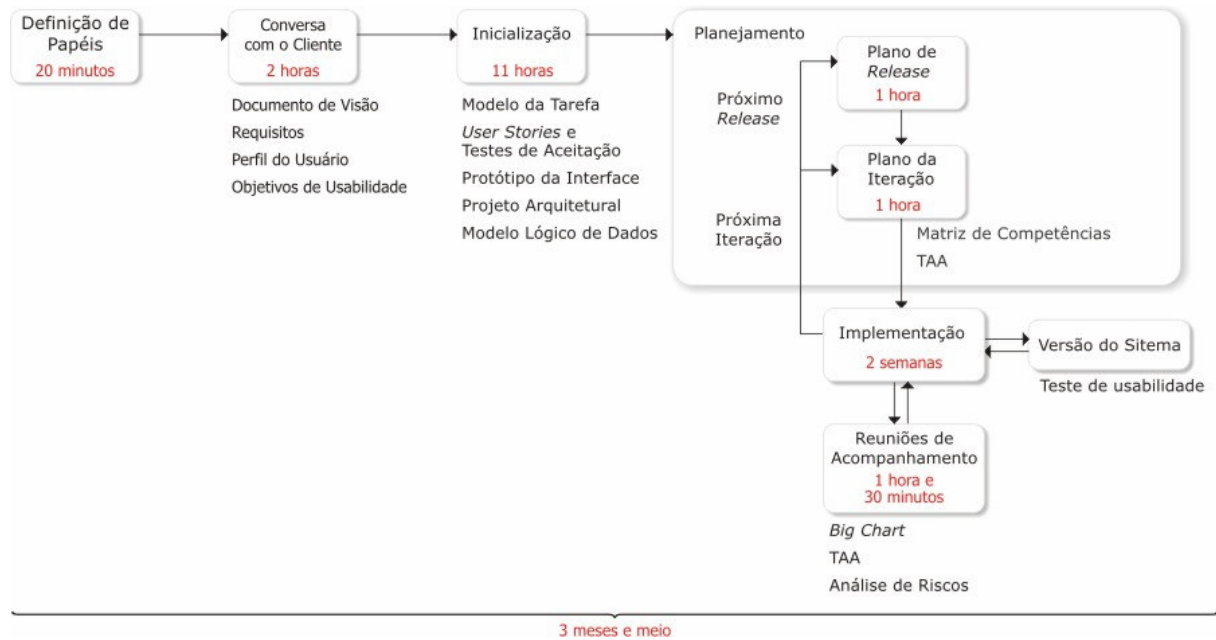
Figura 1 - Síntese do Fluxo do Processo YP

4. Fluxo de Trabalho

1 - **Identificação do Escopo do Problema**;

2 - **Especificação de Papéis**;

3 - **Conversa com o Cliente**:

3.1 - Visão sobre o Processo de Negócio;

3.2 - Levantamento dos Requisitos Funcionais e Não-Funcionais;

3.4 - Levantamento do Perfil do Usuário;

3.5 - Levantamento dos Objetivos de Usabilidade;

4 - **Inicialização**:

4.1 - Modelagem da Tarefa;

4.2 - Levantamento das *User Stories* e Testes de Aceitação;

4.3 - Geração do Protótipo da Interface;

4.4 - Elaboração do Projeto Arquitetural;

4.5 - Geração do Modelo Lógico de Dados;

5 - **Planejamento**:

5.1 - Alocação das *User Stories* nos *releases* (Plano de *Release*);

5.2 - Definição do *release*:

5.2.1 - Alocação das *User Stories* do *release* em questão nas iterações (Plano da Iteração);

5.2.2 - Definição da Iteração:

5.2.2.1 - Desdobramento das User Stories da iteração em atividades;

5.2.2.2 - Construção da Tabela de Alocação de Atividades TAA) e preenchimento com os seguintes dados: atividade, responsável e tempo estimado para realização da mesma;

6 - **Implementação**:

6.1 - Codificação:

6.1.1 - Integração Contínua;

6.1.2 - Boas Práticas de Codificação:

6.1.2.1 - *Design* Simples;

6.1.2.2 - Padrões de Codificação;

6.1.2.3 - Padrões de Projeto;

6.1.2.4 - Refatoramento;

6.1.3 - Propriedade Coletiva de Código;

6.1.4 - Testes Contínuos:

6.1.4.1 - Testes de Unidade;

6.1.4.2 - Testes de Aceitação;

7 - **Finalização da Iteração**:

7.1 - A TAA deve ser concluída (preenchimento dos campos *status* e tempo real do desenvolvimento de cada atividade). Isto deve ser feito na reunião de acompanhamento;

OBS.: Os passos **5.2 até 7.1** devem ser repetidos enquanto houver *releases* ou iterações.

8 - **Versão do Produto**:

8.1 - Realização dos Testes de Usabilidade.

OBS.: Pode ser necessário voltar para o passo 5.2, e prosseguir no fluxo até se obter uma nova Versão do Produto (passo 8), a fim de sanar as falhas encontradas durante a realização dos Testes de Usabilidade.

OBS.: As reuniões de acompanhamento ocorrem ao longo de todo o processo, consistindo da análise do *Big Chart*, Análise de Riscos, e preenchimento da TAA, este último na reunião ao final de cada iteração.

5. Papéis no Processo YP

Ao montar uma equipe de desenvolvimento de software sugere-se uma divisão de tarefas entre os membros da mesma, de forma que cada um assuma um determinado papel no desenvolvimento. Um papel constitui um conjunto de responsabilidades que determinam qual será o comportamento de uma pessoa durante o processo. No YP recomenda-se a presença de cinco papéis: **cliente**, **usuário**, **gerente**, **desenvolvedor** e **testador**.

A alocação dos papéis deve ser feita de acordo com as necessidades e escopo do projeto, levando-se em conta as habilidades e características de personalidade das pessoas envolvidas no processo. A equipe de desenvolvimento deve ser estruturada de forma a se obter a maior produtividade possível.

É essencial que todos os papéis estejam presentes na equipe, pois caso contrário pode ocorrer que algumas etapas importantes do processo sejam esquecidas ou não recebam a atenção necessária. A alocação dos papéis de gerente e testador é dinâmica, significando que um integrante do projeto pode assumir ou se liberar de um destes papéis em determinada etapa do processo.

Um papel não corresponde necessariamente a uma pessoa da equipe, ou seja, uma mesma pessoa pode desempenhar vários papéis simultaneamente. Em equipes pequenas, por exemplo, com três integrantes apenas, isto é geralmente necessário, porém deve-se ter cuidado para que o acúmulo de papéis não leve a uma sobrecarga de responsabilidades de algum membro da equipe.

O papel do cliente é um dos mais importantes, mas nem sempre o cliente possui tempo ou está disposto a participar do desenvolvimento do software. É importante então deixar bem claro para ele quais são as suas responsabilidades e por que sua participação ativa é tão importante para o bom andamento e sucesso do projeto.

PAPÉIS

Cliente – papel desempenhado por quem solicitou o desenvolvimento do software, o que não o condiciona ao uso do produto gerado, ou seja, nem sempre quem requisita o desenvolvimento do software vai ser usuário do mesmo. A presença ativa do cliente no processo é de suma importância,

já que ele sabe quais as funcionalidades do software e conhece as características dos usuários do produto que vai ser gerado. A ausência do cliente pode levar ao fracasso do projeto, com um produto final que não atende a suas necessidades. As principais responsabilidades do cliente são:

- Definir os requisitos do sistema – O cliente deve transmitir o que ele espera que seu sistema seja capaz de fazer, ou seja, quais as funcionalidades e características que o software deve ter para resolver o seu problema;

- Priorizar as funcionalidades – Deve especificar quais partes do sistema são de maior urgência e que devem ser implementadas primeiramente, de forma a permitir que seja feito o planejamento adequado das tarefas a serem realizadas;

- Ajudar na elaboração do plano de *releases* – O cliente deve participar da definição do plano de *releases* junto aos desenvolvedores, de forma que ambas as partes concordem sobre o que deve estar pronto ao final de cada etapa do desenvolvimento;

- Explicitar os testes de aceitação – Testes de aceitação avaliam a eficiência do sistema ao cumprir uma determinada situação. Servem como referencial para que o cliente verifique se o software está realizando a função que lhe foi prometida;

- Identificar o perfil do usuário - Independente do cliente ser usuário do sistema, deve identificar o perfil dos potenciais usuários do sistema. A partir desse perfil inicial, os desenvolvedores podem selecionar uma amostra de usuários para auxiliá-los na validação do protótipo, na avaliação da interface e na realização dos testes de usabilidade.

- Identificar os objetivos de usabilidade - Deve listar, juntamente com o usuário, um conjunto de objetivos mensuráveis, na forma de requisitos de desempenho, por exemplo: reduzir o tempo de execução de uma tarefa específica;

- Validar o protótipo da interface - O cliente e o usuário devem validar o protótipo da interface apresentado pelos desenvolvedores. Só a partir de então se dá inicio a implementação da interface real do sistema;

- Validar o projeto arquitetural - O cliente deve validar o projeto arquitetural elaborado pelo desenvolvedor, apenas após obter um projeto arquitetural consistente os desenvolvedores podem passar a implementação do sistema;

- Ser ativo no processo de desenvolvimento do sistema – O cliente deve estar disponível o máximo de tempo possível para interagir

com os desenvolvedores, seja na definição do sistema, no refinamento de protótipos ou na implantação dos testes de aceitação. Um cliente ausente pode resultar em modelos mal-definidos e sistemas que não atendem aos requisitos do problema.

Usuário – é quem vai de fato utilizar o sistema que será produzido. Muitas vezes, no desenvolvimento de um sistema, o desenvolvedor tende a projetá-lo de acordo com suas próprias preferências, ignorando as preferências do usuário real. Este procedimento geralmente leva à produção de um software de difícil uso e que não satisfaz os desejos do usuário final. A presença do usuário no processo permite que o sistema seja desenvolvido de acordo com seu perfil, suas habilidades e necessidades. Sua participação ativa relembra constantemente o desenvolvedor de que ele está desenvolvendo um sistema que será utilizado por outras pessoas. Vale lembrar que em algumas situações podemos ter que o cliente seja também o usuário do sistema. As principais responsabilidades do usuário são:

- Ajudar a definir os testes de aceitação – O usuário pode ajudar a definir os testes de aceitação para garantir que determinada funcionalidade do sistema que ele considera importante esteja sendo realizada do modo como ele deseja;

- Ajudar a identificar os objetivos de usabilidade - Deve auxiliar o cliente a listar alguns objetivos de usabilidade que devem ser atingidos pelo sistema;

- Validar o protótipo da interface - O usuário e o cliente devem validar o protótipo da interface apresentado pelos desenvolvedores. Só a partir de então se dá início à implementação da interface real do sistema. A validação inicial do protótipo torna a avaliação contínua da interface menos trabalhosa, pois as principais características da interface já estão definidas;

- Avaliar continuamente a interface do sistema – O usuário deve estar constantemente avaliando a interface do sistema, para sugerir alterações que tornem esta interface mais agradável e de fácil utilização. Uma interface mal-projetada e que não leva em consideração as preferências do usuário resulta em um sistema de difícil aprendizado e que o usuário provavelmente evitará utilizar.

Gerente – é o responsável por coordenar as atividades de todos os outros membros da equipe. O gerente deve ser capaz de gerenciar o desenvolvimento e tomar decisões referentes aos riscos e rumos do projeto. As principais competências do gerente são:

- Conduzir os planejamentos e as ações dos desenvolvedores – O gerente deve garantir que os desenvolvedores estejam executando corretamente suas atividades, e no tempo estabelecido, para assegurar o bom andamento do projeto;
- Elaborar o plano de desenvolvimento (release e iteração) – Deve ser realizado junto com os desenvolvedores um planejamento das tarefas a serem realizadas, alocando-se responsáveis para cada tarefa;
- Avaliar sistematicamente os riscos descobertos – A análise de riscos é essencial e deve ser um elemento constante durante o processo. Os riscos devem ser confrontados e sua influência na viabilidade do projeto deve ser considerada;
- Gerenciar configurações – O gerente de projeto deve manter um controle das versões dos artefatos gerados, de forma a garantir que os desenvolvedores estejam sempre utilizando a mesma versão destes artefatos, e que as alterações feitas por um desenvolvedor não produzam artefatos inconsistentes. É aconselhável a utilização de uma ferramenta específica para este controle de versões;
- Coletar e analisar métricas – As métricas são informações que permitem a análise do andamento do projeto pelo gerente, tais como número de linhas de código, n úmero de testes implementados, entre outros. O gerente deve coletar métricas periodicamente junto à equipe de desenvolvimento e utilizá-las na identificação de possíveis riscos e deficiências no projeto;
- Alocar testadores – É responsabilidade do gerente de projeto decidir quem serão os testadores do código produzido por cada um dos desenvolvedores. O código e os testes de unidade produzidos por um desenvolvedor devem ser revisados por outro membro da equipe, a fim de identificar falhas e/ou áreas do software que não foram devidamente testadas;
- Presidir as reuniões de acompanhamento – O gerente deve realizar semanalmente reuniões de acompanhamento com a equipe de desenvolvimento, onde serão avaliados o progresso e as pendências do projeto, utilizando a análise de riscos e o Big Chart. Este

acompanhamento constante é essencial para que se tenha uma gerência eficiente no processo;

- Resolver conflitos internos – O gerente de projeto deve atuar para a manutenção de um ambiente agradável de desenvolvimento, evitando conflitos entre os membros da equipe e outras perturbações;

- Tornar a documentação do projeto sempre atualizada e acessível - A documentação do projeto deve estar disponível e atualizada para que possa ser consultada a qualquer momento pelo time de desenvolvimento. Também é interessante que o cliente possa ter acesso a estas informações, como forma de acompanhar o progresso do projeto.

Desenvolvedor – o papel do desenvolvedor consiste em modelar os requisitos do sistema, e posteriormente produzir um código eficiente e correto que corresponda a tais requisitos. É seu dever verificar a existência de falhas em seu código e, se encontradas, corrigi-las. Para esta verificação devem ser gerados testes de unidade relativos ao código produzido. As principais responsabilidades do desenvolvedor são:

- Levantar os requisitos funcionais e não funcionais junto ao cliente – Deve participar da modelagem do sistema, procurando extrair do cliente todas as informações necessárias para o desenvolvimento do software;

- Auxiliar o gerente na elaboração de um plano de desenvolvimento – O desenvolvedor deve participar do planejamento do desenvolvimento (plano de release e plano de iteração), pois ele tem noção de sua capacidade de geração de código e das dificuldades envolvidas no cumprimento de determinada tarefa. Sua participação no planejamento evita que o gerente superestime as habilidades de seu time de desenvolvimento e atribua prazos irreais à realização das tarefas;

- Analisar e modelar a tarefa - O desenvolvedor deve analisar a tarefa a ser executada pelo usuário com o auxílio do sistema que será desenvolvido. A partir de então, modelar a tarefa em questão de acordo com algum formalismo específico. A modelagem da tarefa orienta a implementação do sistema assim como auxilia na geração da interface do sistema, pois é a partir do modelo da tarefa que são identificadas as relações hierárquicas e de dependência entre as sub-tarefas de uma tarefa mais complexa. Pode ser interessante a participação do usuário do sistema durante a modelagem da tarefa;

- Gerar um protótipo da interface - O desenvolvedor, com as informações do modelo da tarefa, deve gerar um protótipo da interface do sistema. Tal protótipo deve ser simples, na forma de esboço, apenas utilizando lápis e papel, e tem por objetivo mostrar ao cliente e ao usuário, mesmo que de forma grosseira, como será a "cara" do sistema. Esse contato prematuro com a futura interface do sistema possibilita identificar as principais características da interface, facilitando a implementação da mesma. A implementação da interface deve ser iniciada apenas após a validação do cliente e do usuário;

- Identificar os objetivos de usabilidade - O desenvolvedor deve estimular o cliente e o usuário a participar da identificação dos objetivos de usabilidade, com o propósito de tornar o sistema adequado as necessidades dos mesmos;

- Gerar testes de unidade – Cada desenvolvedor deve gerar testes de unidade para o código por ele produzido. O ideal é que tais testes sejam gerados antes da codificação (*Test-Driven Programming* [8]) ou em paralelo com a mesma, o que possibilita um maior entendimento do código e, portanto, a realização de testes mais completos;

- Elaborar o projeto arquitetural - O desenvolvedor deve descrever o funcionamento do sistema a ser desenvolvido num nível alto de abstração, explicitando, além dos relacionamentos entre seus módulos, o relacionamento do sistema com outros sistemas, se for o caso;

- Construir o modelo lógico de dados - Se o sistema a ser desenvolvido envolve um banco de dados, o desenvolvedor deve construir o modelo lógico de dados capaz de representar a estrutura lógica do banco de dados;

- Manter a integração contínua de código – Em projetos onde há mais de um desenvolvedor é necessário que o código gerado por cada um seja integrado com o código produzido pelos demais, para identificar eventuais conflitos. Esta integração deve ocorrer de forma contínua para evitar a perpetuação de conflitos entre partes do software até etapas posteriores, onde estes conflitos serão mais difíceis de serem resolvidos.

Testador – responsável por revisar o código gerado pelos desenvolvedores e por realizar testes de integração do sistema, além de implementar os testes de aceitação definidos pelo cliente. A preparação para a realização dos testes de usabilidade também é de responsabilidade do testador.

- Efetuar revisões sobre o código de outro desenvolvedor – O testador deve revisar o código produzido pelos desenvolvedores, com o propósito de identificar possíveis falhas e necessidades de refatoramento, garantindo assim a qualidade do código gerado;
- Elaborar testes sobre o código de outro desenvolvedor – O testador deve analisar os testes de unidade feitos pelos desenvolvedores, identificar partes do código não testadas e refinar os testes para que cubram estas partes. O testador do código não deve ser a mesma pessoa que desenvolveu o código, pois o desenvolvedor tende a não enxergar as deficiências de seu próprio código;
- Refatoramento de código - Cabe ao testador fazer melhorias no código dos desenvolvedores a fim de deixar o código cada vez mais simples, legível, flexível e claro. O refatoramento não altera as funcionalidades implementadas. Os testes devem ser executados com sucesso antes e depois do refatoramento, garantindo que as funcionalidades permanecem intactas;
- Gerar testes de aceitação – Antes de cada entrega de uma versão funcional os testadores devem implementar os testes de aceitação propostos pelo cliente.
- Elaborar o material para realização dos testes de usabilidade – Uma vez que se tem uma versão do sistema, deve ser realizada uma bateria de testes de usabilidade. Para tanto é necessário que os testadores recrute usuários (de teste), elabore o roteiro de atividades e o questionário pós-teste.

Resumo

Papéis:

- Cliente;
- Usuário;
- Gerente;
- Desenvolvedor;
- Testador.

Responsabilidades do Cliente:

- Definir os requisitos do sistema;
- Priorizar as funcionalidades;
- Ajudar a elaborar o plano de release;
- Explicitar os testes de aceitação;
- Identificar o perfil do usuário;
- Identificar os objetivos de usabilidade;
- Validar o protótipo da interface;
- Validar o projeto arquitetural;
- Ser ativo no processo de desenvolvimento de sistema.

Responsabilidades do Usuário:

- Ajudar a definir os testes de aceitação;
- Ajudar a identificar os objetivos de usabilidade;
- Validar o protótipo da interface;
- Avaliar continuamente a interface do sistema.

Responsabilidades do Gerente:

- Conduzir os planejamentos e as ações dos desenvolvedores;
- Elaborar o plano de desenvolvimento (release e iteração);
- Avaliar sistematicamente os riscos descobertos;
- Gerenciar configurações;
- Coletar e analisar métricas;
- Alocar testadores;
- Presidir as reuniões de acompanhamento;
- Resolver conflitos internos;
- Tornar a documentação do projeto sempre atualizada e acessível.

Responsabilidades do Desenvolvedor:

- Levantar os requisitos funcionais e não funcionais junto ao cliente;
- Auxiliar o gerente na elaboração de um plano de desenvolvimento (release e iteração);
- Analisar o modelar a tarefa;
- Gerar um protótipo da interface;
- Identificar os objetivos de usabilidade;
- Gerar testes de unidade;
- Elaborar um projeto arquitetural;
- Gerar o modelo de dados;
- Manter a integração contínua de código.

Responsabilidades do Testador:

- Efetuar revisões sobre o código de outro desenvolvedor;
- Elaborar testes sobre o código de outro desenvolvedor;
- Refatoramento de código;
- Gerar testes de aceitação;
- Elaborar o material para realização dos testes de usabilidade.

Representação dos Papéis no Fluxo do YP
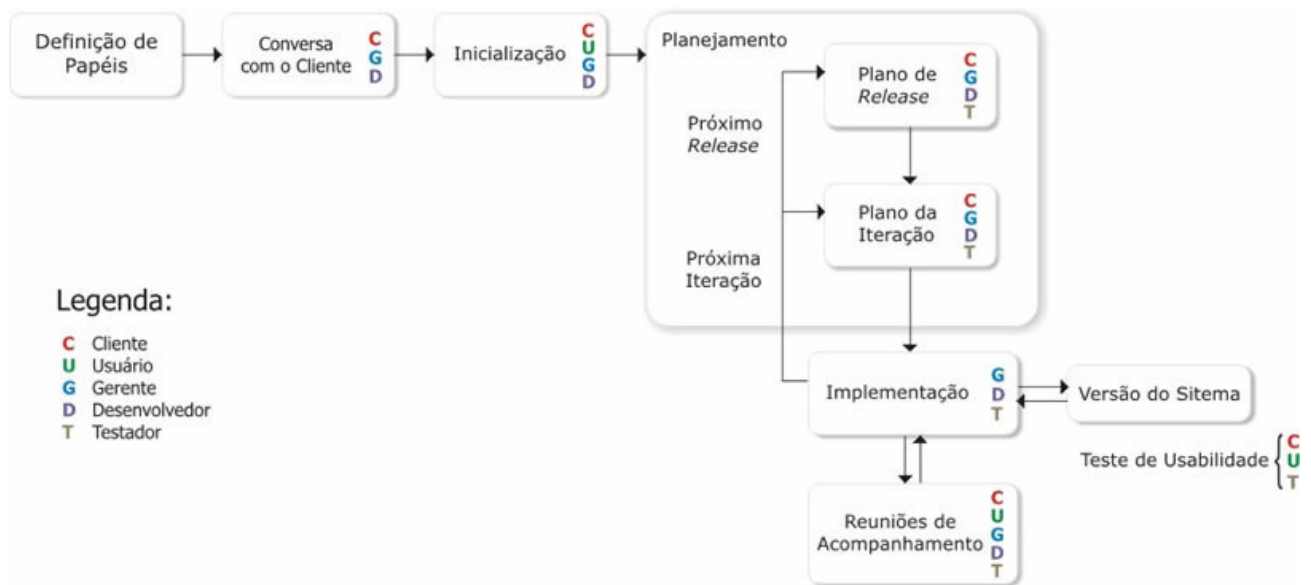
Figura 2 - Representação dos Papéis no Fluxo do YP

6. Primeira Conversa com o Cliente

A primeira conversa com o cliente consiste em uma coleta inicial de informações sobre o sistema que será desenvolvido. A partir dessa conversa o cliente e os desenvolvedores passam a ter um entendimento comum a respeito do sistema. O desenvolvedor deve tentar extrair do cliente a maior quantidade de informação possível sobre o sistema solicitado, ou seja, deve buscar por informações relacionadas às principais funcionalidades do sistema (**requisitos funcionais**), aos **requisitos não-funcionais** mais importantes, ao **perfil do usuário**, aos **objetivos de usabilidade**, aos testes de aceitação, entre outros. Para tanto, faz-se necessário que o desenvolvedor esteja preparado para o grande fluxo de informação proveniente dessa conversa, logo, sugere-se que o desenvolvedor esteja ciente do que devem fazer antes, durante de depois da reunião. É essencial que o cliente perceba sua importância no processo de desenvolvimento do sistema, assim cabe ao desenvolvedor fazê-lo entender a necessidade de sua participação ativa.

O que deve ser feito antes:

- Entender o escopo do problema: os desenvolvedores devem se informar sobre o domínio do problema;
- Elaborar um roteiro de perguntas: este roteiro tem o objetivo de ajudar a equipe a levantar as informações mais importantes.

Como o desenvolvedor deve ser portar durante a conversa:

- Tornar a reunião produtiva: procurar extrair o máximo de informações em um tempo mínimo (depende da disponibilidade do cliente) e incentivar a participação do cliente para que este se sinta parceiro do processo;
- Usar uma linguagem simples: evitar o uso de termos técnicos, para que não haja dúvidas nem interpretações distintas por nenhuma das partes;
- Fazer uso de analogias: analogias podem ser usadas para possibilitar o maior entendimento entre o cliente e a equipe de desenvolvimento;

- Informar ao cliente o seu papel: esta informação deve ser dada de forma sutil, mostrando ao cliente a necessidade de sua participação para o sucesso do desenvolvimento do software;
- Agendar reuniões periódicas com o cliente: antecipar uma agenda de reuniões com o cliente é importante. O cliente deve se sentir parte integrante do projeto com a motivação de garantir que as suas necessidades sejam satisfeitas.

O que a equipe deve estar sabendo ao final da conversa:

- Ter uma idéia bem definida do escopo do problema: um passo importante para elaboração do documento de visão;
- Conhecer o perfil dos usuários do sistema e suas necessidades;
- Saber quais os são requisitos funcionais e não funcionais do sistema;
- Listar os riscos do projeto: a equipe de desenvolvimento deve saber quais as dificuldades mais críticas do projeto e os riscos envolvidos. Alguns desses riscos podem levar o projeto ao fracasso, expondo a necessidade de se analisar sua viabilidade.

Resumo

O que é?

É a primeira conversa entre o cliente e os desenvolvedores do sistema.

Para que serve?

Tem por objetivo fazer com que o cliente o os desenvolvedores tenham uma idéia comum a respeito do sistema que será desenvolvido.

Quem faz?

O cliente e os desenvolvedores.

O que os desenvolvedores devem fazer?

- **Antes:**
  - Entender o escopo do problema;
  - Elaborar um roteiro de perguntas.
- **Durante:**
  - Tornar a reunião produtiva;
  - Usar uma linguagem simples;
  - Fazer uso de analogias;
  - Informar ao cliente o seu papel;
  - Agendar reuniões periódicas com o cliente.
- **Depois:**
  - Ter uma idéia bem definida do escopo do problema;
  - Conhecer o perfil dos usuários do sistema e suas necessidades;
  - Saber quais são os requisitos funcionais e não funcionais do sistema;
  - Listar os riscos do projeto.

Duração: Cerca de 1 hora.

7. Documento de Visão

Concluída a conversa inicial com o cliente, a primeira tarefa a ser feita passa a ser a elaboração de um documento de visão, contendo as idéias gerais sobre o que o sistema se propõe a fazer de forma que possa apoiar os processo de negócios do cliente.

Neste momento, o desenvolvedor possui as informações fornecidas pelo cliente e já tem uma idéia bem amadurecida acerca do sistema a ser desenvolvido. Por isso, esta é a ocasião ideal para se pensar no sistema como um todo, documentando suas características gerais.

O documento de visão deve ser feito utilizando uma linguagem de fácil entendimento para o cliente, uma vez que deve ser usado como elo de comunicação entre este e o desenvolvedor. Uma vez produzido, o documento deve ser validado junto ao cliente no intuito de se descobrir se a visão de negócio do sistema, definida pela equipe de desenvolvimento, realmente está correta. Sendo validada, a visão servirá como base para a atividade de planejamento e poderá sempre ser consultada pela equipe quando houver qualquer dúvida sobre o que foi acertado com o cliente no momento inicial. Analisando desta maneira, nota-se que o documento de visão funciona como um contrato entre cliente e desenvolvedor, de forma que a equipe de desenvolvimento não deve ”imaginar” coisas novas incoerentes com o que especifica este “contrato”.

Entretanto, o documento de visão não deve ser visto como o ponto final sobre o que deve ser feito. Ocorrendo mudanças de requisitos durante o andamento do processo, a visão deve refletir essas modificações. Porém, o desenvolvedor deve estar sempre atento, pois nem sempre uma mudança de requisitos se reflete em mudanças na visão de negócio do cliente.

É importante que o documento de visão seja objetivo e traga uma quantidade mínima de informações. É essencial se começar com uma breve descrição do problema, na intenção de se tornar mais fácil descrever os processos de negócios do cliente e como o sistema se propõe a resolver. Se o problema possui regras de negócio e termos pouco familiares a equipe, talvez seja necessário se construir um glossário. A má compreensão de idéias acerca do problema pode vir a colocar o projeto por água abaixo.

Deve-se aproveitar o documento de visão para definir quem são os usuários finais do sistema. A partir de informações iniciais sobre os usuários é possível traçar um perfil para o mesmo, e daí selecionar alguns usuários potencias para participarem de fases posteriores do processo, em atividades tais como: validação do protótipo da interface, testes de

usabilidade e outras. A identificação desses usuários e inserção dos mesmos no processo de desenvolvimento do sistema é de extrema importância, pois é através deles que serão abordadas as principais questões relacionadas à usabilidade do sistema.

Deve-se então, definir as características gerais do produto, bem como os requisitos funcionais e não funcionais. É sempre bom ter em mente que este é um documento com informações de alto nível. Os detalhes de baixo nível sobre os requisitos serão oportunamente definidos ao longo a etapa de inicialização. Na definição das características, é recomendável o uso de analogias, de forma que a compreensão da equipe seja maximizada. Analogias são bastante úteis quando o domínio do problema não é especialidade da equipe, ajudando todo o grupo a ter uma idéia única e correta no que diz respeito ao negócio. Analogias também podem ser utilizadas durante a descrição do problema.

Para fechar o grupo de elementos essenciais que compõem o documento de visão, sugere-se a inclusão das limitações do projeto e possíveis riscos. A equipe deve ter consciência dos fatores que podem causar desvios indesejáveis no desenvolvimento. Destaca-se mais uma relevância do documento de visão: servir para que se possa analisar se o projeto é realmente viável.

Resumo

O que é?

É um artefato que define em alto nível o domínio do problema, mapeando os processos de negócios do cliente a serem suportados pelo sistema. Uma maneira direta de comunicar o que se pretende no projeto para todos os envolvidos.

Para que serve?

- Avaliar se a equipe de desenvolvimento entendeu corretamente o domínio do problema descrito pelo cliente;
- Sinalizar se houve mudanças muito grandes no projeto durante o processo;
- Fazer com que o desenvolvedor pense no sistema como um todo, inserido no cenário de negócio do cliente, antes de começar a desenvolvê-lo, evitando alterações constantes e minimizando alguns tipos de riscos;

- Avaliar se o desenvolvimento do projeto é viável;
- Após ser validado pelo cliente passa a ser um contrato entre este e o desenvolvedor;

Quem faz?

Os desenvolvedores, a partir de conversas com o cliente sobre o domínio do problema.

Quando deve ser gerado?

Logo após a primeira conversa com o cliente.

O que o documento deve responder:

- Quais os principais termos envolvidos no escopo do problema (glossário)?
- Qual(is) o(s) problema(s) a ser(em) resolvido(s)? Como tal(is) problema(s) afeta(m) o processo de negócio do cliente?
- Quem são os usuários e os demais envolvidos com o projeto? E quais as suas necessidades?
- Quais são as características do produto?
- Quais os requisitos funcionais?
- Quais são os requisitos não funcionais?
- Quais são as limitações do projeto?
- Quais são os riscos do projeto?

Duração: Cerca de 30 minutos

7.1. Requisitos

Resumo

O que são?

São objetivos e/ou restrições, listados por clientes e usuários, que definem as características e funções de um sistema. Tais requisitos podem ser funcionais ou não-funcionais. Os Requisitos Funcionais descrevem as várias funções que o software deve possuir, como por exemplo: "o sistema deve emitir relatórios de vendas diárias automaticamente". Os Requisitos Não-Funcionais definem as propriedades e restrições, tais como: manutenibilidade, usabilidade, desempenho. Um exemplo seria: "o tempo de resposta do sistema não deve ser superior a 30 segundos". Para que os requisitos possam ser verificados e validados é necessário que os mesmos devem possuir uma descrição correta, completa, não ambígua e verificável. Todos os requisitos do sistema devem estar presentes no documento de visão.

Para que serve?

Para especificar as características do sistema, ou seja, o que ele deve fazer ou como ele deve operar.

Quem faz?

O cliente e o desenvolvedor

Quando faz?

Durante a primeira conversa com o cliente.

Duração: Cerca de 30 minutos.

7.2. Perfil do Usuário

Resumo

O que é?

É um conjunto de informações relacionadas às características de potenciais usuários do sistema. Tais características devem revelar as habilidades, as limitações, as preferências, os interesses dos usuários, assim como o conhecimento prévio dos mesmos com relação a tarefa a ser realizada e ao uso de sistemas computacionais. As características que devem ser identificadas durante o levantamento do perfil do usuário devem ser relevantes para o desenvolvimento do sistema. O perfil do usuário pode ser descrito no documento de visão.

Para que serve?

Para que a equipe de desenvolvimento possa selecionar usuários a fim inseri-los no processo de desenvolvimento do sistema. De posse desse artefato, o papel de usuário pode ser bem representado.

Quem faz?

O cliente

Quando de faz?

Durante a primeira conversa com o cliente.

Duração: Cerca de 20 minutos.

Lista de Características Relevantes:

Algumas das características citadas abaixo podem ser relevantes para o desenvolvimento do sistema em questão, portanto devem ser exploradas

na primeira conversa com o cliente, a fim de identificar o perfil do usuário. No entanto, de acordo com as especificações do software, outros aspectos importantes podem surgir. Sendo assim, as características relevantes não se limitam, apenas, às listadas.

- Sexo;
- Canhoto, destro ou ambidestro;
- Uso de lentes corretivas;
- Faixa etária;
- Experiência prévia no uso de sistemas computacionais;
- Tempo de uso de sistemas computacionais;
- Freqüência de uso de sistemas computacionais;
- Plataforma computacional que utiliza com maior freqüência;
- Conhecimento prévio de uma outra versão do sistema que está sendo desenvolvido (melhorado);
- Freqüência de uso de tal versão;
- Familiaridade com a língua inglesa;
- etc.

7.3. Objetivos de Usabilidade

Resumo

O que é?

É um conjunto de metas de usabilidade, mensuráveis, que devem ser alcançados pelo sistema. Geralmente se referem à eficácia, eficiência, segurança, aprendizado (*learnability*), memorização (*memorability*) do sistema. Onde:

- Eficácia: está relacionada com a forma de realização da tarefa e a disposição das informações necessárias para a realização da mesma;
- Eficiência: refere-se ao auxílio prestado ao usuário, pelo sistema, na realização de suas tarefas;
- Segurança: consiste em proteger os usuários de condições perigosas e situações indesejáveis, ou seja, prevenir que os usuários cometam erros graves;
- *Learnability*: diz respeito à fácilidade de aprender a usar o sistema;
- *Memorability*: considera a facilidade que o usuário tem de lembrar como utilizar um sistema, depois de já ter aprendido como fazê-lo.

Para que serve?

Os objetivos de usabilidade transformam-se em critérios que permitem avaliar a usabilidade do sistema a partir do desempenho do usuário. Ou seja, a eficiência do sistema, por exemplo, pode ser mensurada a partir do tempo que o usuário leva para executar uma determinada tarefa.

Quem faz?

O cliente e o usuário

Duração: Cerca de 30 minutos.

8. Inicialização

Logo após a equipe de desenvolvimento ter uma idéia geral sobre o problema a ser resolvido, devem ser iniciadas algumas atividades de análise do sistema.

O objetivo é fazer um **modelo da tarefa** para que sirva de base construção do **protótipo da interface** que tem a função de para receber o *feedback* do cliente nas fases iniciais de implementação e listar todas as ***User Stories*** do sistema (e seus respectivos **Testes de Aceitação**) a fim de desenvolver uma **arquitetura**, um **modelo lógico de dados** capazes de acomodar mudanças e que sejam estáveis o suficiente para que riscos sejam minimizados.

Resumo

O que é?

É um conjunto de 5 atividades: (1) **Modelagem da Tarefa**; (2) definição das ***User Stories*** do sistema, com seus respectivos **Testes de Aceitação**, (3) geração do **Protótipo da Interface**, (4) elaboração do **Projeto Arquitetural**; e por fim, (5) construção de um **Modelo Lógico de Dados**, caso exista um banco de dados envolvido.

Para que serve?

Para que os desenvolvedores possam ver o futuro sistema de acordo com aspectos de diversas naturezas. Ou seja, passam e entender: como a tarefa funciona, quais são as funções do sistema (*User Stories*), como se dá a interação do usuário com o sistema, como os módulos do sistema se relacionam entre si e com módulo ou sistemas externos, como o sistema deve ser testado, e como o sistema deve ser estruturado logicamente.

Quem faz?

- Modelagem da Tarefa - Desenvolvedor
- *User Stories* e Testes de Aceitação- Cliente e Desenvolvedor
- Protótipo da Interface - Desenvolvedor*

- Projeto Arquitetural - Desenvolvedor**
- Modelo Lógico de Dados - Desenvolvedor

* Precisa ser validado pelo Cliente e pelo Usuário
** Precisa ser validado pelo Cliente

Duração: Cerca de 11 horas.

Obs.: Vale a pena investir tempo aqui, pois com um bom entendimento do sistema, sob diversos aspectos, a fase de implementação apresenta menos problemas.

8.1. Modelo da Tarefa

Resumo

O que é?

É uma representação, hierárquica, de como a tarefa é realizada pelo usuário. A construção do modelo da tarefa é precedida pela análise da tarefa, que consiste no estudo da tarefa a fim de melhorar a compreensão dos desenvolvedores com relação a natureza da tarefa, as partes que a compõe, e a ordem com que devem ser realizadas. Durante a análise da tarefa aspectos relacionados às características da tarefa, ao perfil do usuário e aos objetivos de usabilidade devem ser considerados.

A análise da tarefa capacita os desenvolvedores a modelar a tarefa, decompondo a tarefa raiz (mais alto nível de abstração, por exemplo: utilizar o sistema X) em sub-tarefas, sucessivamente, até atingir as ações elementares, e em seguida definindo a seqüência de realização das mesmas.

O modelo da tarefa pode ser construído de acordo com vários formalismos, no entanto, o YP sugere o uso do TAOS (*Task and Action Oriented System*)[13].

Para que serve?

- Para melhorar o entendimento, por parte da equipe de desenvolvimento, de como o usuário deve interagir com a interface do sistema para executar uma tarefa;
- Auxiliar na construção do protótipo da interface;
- Suportar a elaboração do roteiro de atividades para a realização dos testes de usabilidade.

Quem faz?

O Desenvolvedor, podendo ser auxiliado pelo Cliente ou Usuário.

Duração: Cerca de 2 horas.

8.1.1. Formalismo TAOS (Task and Action Oriented System)

O TAOS (*Task and Action Oriented System*) visou, inicialmente, a construção de Sistemas Baseados em Conhecimento (SBC), tendo sido validado no domínio da biologia molecular – Inteligência Artificial. Porém, como todos os requisitos exigidos pela modelagem da tarefa eram contemplados por este formalismo, o seu uso se consagrou como formalismo para aquisição e representação do conhecimento baseado na modelagem do domínio.

Este formalismo é composto por dois módulos: TAME (*Task and Action Modeling Environment*) e *Graph*. O primeiro define a linguagem para a modelagem do conhecimento na forma de uma taxonomia, fazendo a validação de completude e coerência do conhecimento. O segundo possibilita a visualização do processo de modelagem.

O funcionamento do TAOS pode ser acompanhado a partir da figura abaixo, onde: uma **Tarefa** tem por objetivo sair de uma situação inicial, um ponto de partida, e alcançar um objetivo, situação final. Para que este objetivo seja alcançado, a Tarefa é decomposta e **Ações** e **Sub-Tarefas**, que têm seus relacionamentos gerenciados a partir de **Métodos**. Cada **Ação** deve ser executada por um **Agente** com o auxílio de um **Instrumento**.

As tarefas são decompostas em sub-tarefas e ações elementares, formando uma árvore hierárquica. Cada nodo da árvore é identificado unicamente, e tem sua ordem e relacionamentos estabeleciodos pelo formalismo, a partir dos operadores abaixo:

- SEQ: sub-tarefas e/ou ações devem ser executadas em seqüência;
- OR: pelo menos uma das sub-tarefa e/ou ações deve ser executada;
- XOR: apenas uma das sub-tarefa e/ou ações deve ser executada;
- AND: todas as sub-tarefa e/ou ações devem ser executadas, não importando a ordem;
- SIM: sub-tarefa e/ou ações podem ser executadas simultaneamente de forma independente;
- PAR: sub-tarefa e/ou ações podem ser executadas concorrentemente, ou seja, com pontos de sincronização.
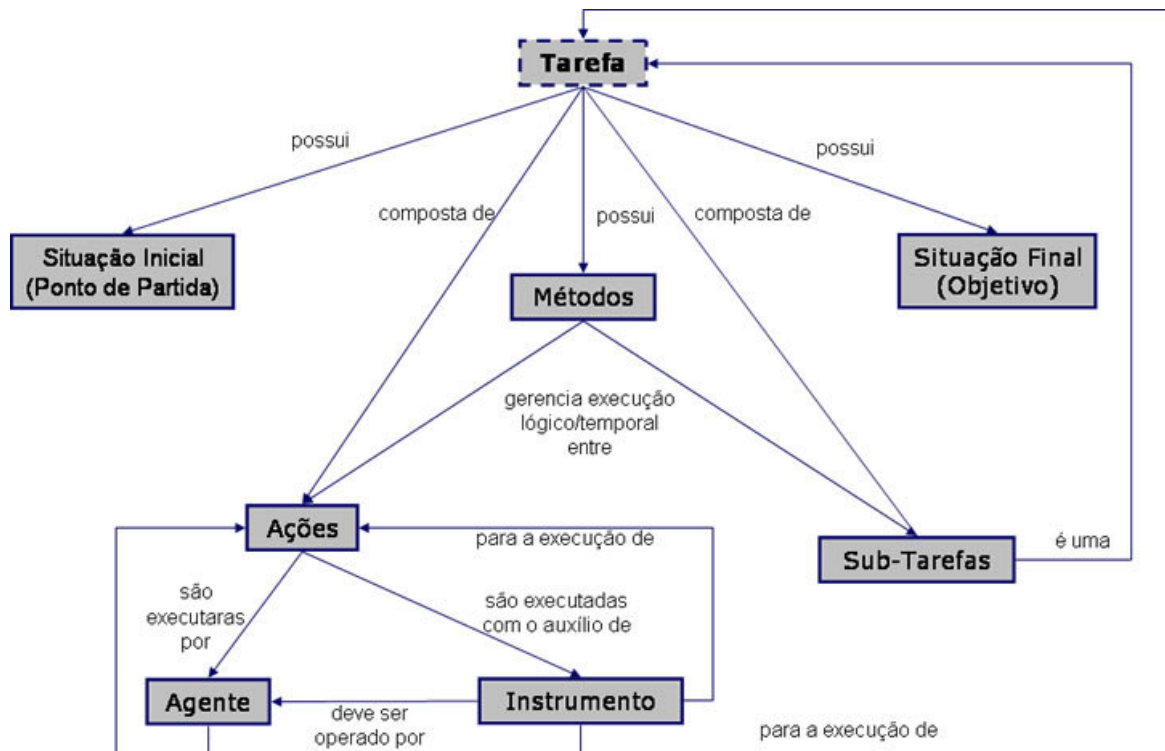
Figura 3 - Representação dos Elementos Presentes no Formalismo TAOS

Definindo os elementos da imagem, temos:

- Tarefa: É uma seqüência de sub-tarefas/ações organizadas de maneira a atingir um objetivo;
- Ação: Descreve uma ação elementar;
- Método: Define a estratégia para executar uma tarefa (Expressões que permitem estabelecer relações temporais e/ou lógicas entre sub-tarefas e/ou ações que compõem uma tarefa) utilizando os operados (SEQ, OR, XOR, AND, SIM e PAR);
- Agente: Define um tipo de objeto habilitado a executar uma ação;
- Instrumento: Define um objeto (ferramenta) que é utilizada pelos agentes para executar ações.

8.2. User Stories e Testes de Aceitação

Resumo

O que é?

São especificações feitas pelo cliente a fim de definir o que o sistema deve ter/fazer. É com base nas *User Stories* que as funções do sistema são implementadas. Recomenda-se que a maior parte das *User Stories* sejam aqui definidas nessa fase do processo, porém durante o andamento do projeto outras poderão surgir, assim como as existentes podem ser modificadas ou até mesmo eliminadas. Sugere-se que se a *User Story* for muito grande, divida-a; se for muito pequena, agrupe-a com outras.

É interessante levar em consideração o tempo de desenvolvimento de cada *User Story*, essa estimativa inicial pode ou não ser confirmada no planejamento da iteração, quando as *User Stories* são quebradas em atividades menores e é feita uma estimativa para a realização de cada uma das atividades. No entanto, com essa estimativa inicial é possível confrontar o tempo disponível para o projeto e o exigido para a implementação de todas as *User Stories*.

Para cada ***User Story*** deve estar associada a pelo menos um Teste de Aceitação (mais detalhes no documento sobre Testes).

Para que serve?

- Facilitar o entendimento da equipe de desenvolvimento com relação as funções do sistema, uma vez que essas aparecem contextualizadas em uma estória de uso específica;
- Auxiliar o cliente a elaborar os testes de aceitação;
- Orientar o cliente na identificação da priorização da ordem de implementação das funções do sistema;
- Ajudar a definir o escopo do projeto, pois, caso o tempo inicial estimado para a realização de todas as *User Stories* ultrapassar o tempo disponível para o projeto, o escopo do mesmo terá que ser redefinido, uma vez que para YP o tempo é fixo;
- Auxiliar o desenvolvedor a analisar o modelar a tarefa.

Quem faz?

- Definição e Priorização das *User Stories*: cliente;
- Elaboração dos Testes de Aceitação: cliente;
- Implementação das *User Stories*: desenvolvedor;
- Execução dos Testes de Aceitação: testador.

Duração: Cerca de 3 horas.

8.3. Protótipo da Interface

Resumo

O que é?

É uma representação gráfica, não necessariamente funcional, de um sistema que ainda não foi implementado. O YP preconiza a construção de protótipos simples (esboço), que possam ser construídos com baixo investimento de tempo e recurso sem requerer qualquer habilidade técnica.

Tais protótipos são bastante eficientes nas fases iniciais do processo de desenvolvimento de software, pois, a representação gráfica possibilita a comunicação através de uma linguagem comum a todos os envolvidos no desenvolvimento do sistema. Desta maneira existe uma maior facilidade na compreensão de conceitos do contexto de uso, de aspectos de navegação, interação, ou detalhes operacionais e estéticos do sistema.

Para que serve?

- Facilitar o entendimento da semântica e do contexto de uso do sistema;
- Estruturar o fluxo de interação entre o sistema e o usuário;
- Explorar idéias de design antes de investir tempo e recursos na implementação da interface do usuário.

Quem faz?

O Desenvolvedor a partir do modelo da tarefa, do perfil do usuário dos requisitos levantados pelo cliente.

Duração: Cerca de 2 horas.

8.4. Projeto Arquitetural

Este artefato tem como objetivo descrever o funcionamento do sistema num alto nível de abstração. Ele é útil quando se deseja explicitar como as partes do sistema a serem desenvolvidas interagem entre si ou com outros sistemas.

Em geral, o projeto arquitetural contém informações que sofrem pouca ou nenhuma alteração ao longo do processo, a menos que grandes mudanças aconteçam. Os detalhes de projeto ou implementação não devem ser descritos.

O YP sugere que para compor o projeto arquitetural seja construído um diagrama contendo a estrutura da arquitetura, de forma que as possíveis dependências e relacionamentos, entre o sistema que está sendo desenvolvido e demais sistemas, sejam explicitadas. Esse diagrama, se necessário, pode vir acompanhado de um texto que explique de forma objetiva os aspectos de maior relevância do sistema.

O projeto arquitetural deve ser elaborado pela equipe de desenvolvimento, a qual deve estar ciente dos pontos críticos do sistema, das dependências existentes e dos requisitos levantados (particularmente os não funcionais). Depois da elaboração, a arquitetura deve ser aprovada pelo cliente.

Esse artefato deve ser produzido durante a fase de inicialização, pois os conhecimentos necessários estão bem amadurecidos. Sugere-se, no entanto, que a sua construção ocorra após a definição e priorização das *User Stories*. Não é necessário que as *User Stories* estejam divididas em atividades.

Todos os desenvolvedores devem participar da elaboração deste artefato. O cliente é o responsável por validar a arquitetura definida. Só após a validação do cliente é que este documento é considerado pronto. Este documento pode ser mudado, tendo que ser validado novamente pelo cliente.

RESUMO

O que é?

É um artefato que define em alto nível o domínio e o funcionamento do sistema, mostrando de forma clara possíveis dependências e/ou

relacionamentos internos entre os seus módulos, ou externos, entre o sistema a ser desenvolvido e sistemas externos, se for o caso.

Para que serve?

- Mostrar a estrutura do sistema;
- Mostrar dependências e/ou relacionamentos existentes entre os módulos do sistema;

Quem faz?

Os desenvolvedores, a partir de conversas com o cliente e artefatos gerados anteriormente, como: o documento de visão, requisitos não-funcionais, a lista de *User Stories* e o Modelo da Tarefa. Este documento deve ser validado pelo cliente.

Quando deve ser gerado?

Durante a inicialização, depois que as *User Stories* e o Modelo da Tarefa estão definidos. Para a sua elaboração não é necessário que as *User Stories* estejam divididas em atividades, no entanto é importante que já tenham sido priorizadas pelo cliente.

Duração: Cerca de 1 hora e 30 minutos.

O que o documento deve responder?

- Como acontece o funcionamento do sistema?
- Quais os principais módulos do sistema?
- Quais as relações e/ou dependências entre eles?
- Quais as relações e/ou dependências em relação a outros sistemas?

9. Planejamento (Release e Iteração)

Concluídas as atividades de inicialização, deve ser iniciado o planejamento do projeto. Primeiramente, a equipe deve estar ciente do tempo disponível para o desenvolvimento do projeto, e a partir de então, com o cliente, definir o número de *releases* e iterações necessárias para a conclusão do mesmo.

Tratando-se do escopo de uma disciplina, o YP recomenda 3 *releases* por semestre letivo, cada uma contendo duas iterações de 2 semanas cada – três meses no total.

Como o YP propõe pequenos *releases* na tentativa de garantir uma maior interação entre a equipe de desenvolvimento e o cliente, sendo assm, as *User Stories* alocadas em cada *release* só podem ser consideradas como finalizadas após a realização dos testes de aceitação. Um benefício desta prática é a redução do impacto das mudanças de requisitos.

Considerando que cada *User Story* alocada seja finalizada após a aceitação do cliente, pode-se considerar que tudo o que esteja implementado funcione satisfazendo a especificação do projeto. Mesmo que haja erros na implementação de algum requisito, o impacto é bem menor do que realizar uma implementação com release mais longo.

É interessante gerar uma matriz de competências, mapeando as habilidades de cada membro da equipe de desenvolvimento, assim como o número de horas por semana que cada um poderá dedicar ao projeto. O uso dessa matriz de facilitará bastante a alocação das atividades, descrita mais à frente.

Considerando que todas as *User Stories* já foram definidas, deve ser iniciado o planejamento de *release*.

Plano de *Release*

O plano de *release* consiste em definir o período dos *releases* e a partir de então dividi-lo em iterações. Em seguida, as *User Stories* e os testes de aceitação associados às mesmas devem ser distribuídas nas iterações do *release* em questão. Vale ressaltar que os testes de aceitação devem ser definidos nesta etapa do desenvolvimento e não durante o processo. Cada *User Story* deve ter pelo menos um teste de aceitação.

Tal associação deve levar em consideração a priorização feita pelo cliente, no entanto, é possível que a equipe de desenvolvimento necessite alertar o cliente sobre *User Stories* que tecnicamente devem ser

primeiramente implementadas. Por exemplo, aconselha-se que *User Stories* de maior risco para o projeto sejam atacadas no início, pois o risco de um projeto ser abortado depois de muito tempo de desenvolvimento é minimizado.

A alocação das *User Stories* deve ser feita apenas para o próximo *release*, pois planejar *releases* à frente significa que mudanças ou aparecimento de novos requisitos podem vir a invalidar um planejamento de atividades ainda distantes.

É relevante observar que, durante a alocação das *User Stories*, o tempo de um *release* ou iteração é fixo. O escopo, ou seja, a quantidade de tarefas e/ou *User Stories* que serão implementadas, é que deve variar.

Plano da Iteração

O plano da iteração consiste em quebrar as *User Stories* em atividades menores (caso necessário) e em seguida alocar os membros da equipe de desenvolvimento como responsáveis pela realização de cada atividade. A alocação da tarefa ao desenvolvedor pode ser feita com o auxílio da Matriz de Competências, uma tabela simples que informa quais as habilidades dos membros da equipe de desenvolvimento e o tempo que cada um disponibiliza para o projeto.

Após a atribuição das atividades aos desenvolvedores, cada desenvolvedor deve estimar o tempo para realização de cada atividade, respeitando o período estabelecido para a iteração. Ao especificar o tempo de desenvolvimento das atividades, talvez seja interessante estimar "por cima", resultando assim num tempo de contingência.

O YP sugere fazer uso da Tabela de Alocação de Atividades (TAA), para listar todas as atividades da iteração, especificando o tempo de desenvolvimento e o responsável por cada uma delas. É importante perceber que neste momento a TAA não é completamente construída, pois campos como tempo real para conclusão da atividade e *status* só serão preenchidos no final da iteração.Assim como no planejamento de *release*, o planejamento da próxima iteração só se dá ao término da anterior.

Dicas

- Ao elaborar o planejamento das iterações ou dos *releases*, deve-se levar em consideração as atividades e/ou *User Stories* não concluídas;
- Durante todo o planejamento, riscos devem ser identificados e analisados. Em geral, os riscos são realmente percebidos no momento da implementação;

- Se houver necessidade de rever o escopo do projeto, isto deve ser feito. Em nenhuma hipótese o tempo planejado para cada iteração deve ser alterado, apenas o escopo do projeto, caso seja necessário um ajuste;
- Percebe-se então que o planejamento de *release* termina juntamente com o planejamento da última iteração deste *release*;
- Um cronograma da iteração, especificando a data de início e término de cada atividade pode ser construído a fim de melhor gerenciar o andamento das atividades;
- É extremamente importante que o gerente disponibilize todo o planejamento em um local onde a equipe de desenvolvimento e o cliente possam acessar, como por exemplo, um site.

Resumo

O que é?

O planejamento que descreve de forma clara e objetiva um programa das atividades que serão realizadas ao longo do desenvolvimento do sistema. No YP a fase de planejamento é composta por dois planejamentos, o de *release* e o da iteração, no qual existem 3 *releases,* cada um contendo 2 iterações de 2 semanas.

Para que serve?

É útil na organização, no planejamento a curto e em longo prazo e no cumprimento das atividades do projeto. Auxilia o gerente a avaliar o desempenho de toda a equipe de desenvolvimento quanto ao andamento do projeto.

Quem faz?

O cliente e toda a equipe de desenvolvimento participam do planejamento do *release* e da iteração. Mas é responsabilidade do gerente documentar e disponibilizar de forma clara estes planos.

Como se faz?

- Plano de *release* consiste em 3 *releases;*

- Cada *release* contém 2 iterações;
- Cada iteração está associada a um conjunto de *User Stories* (de acordo com a priorização do cliente) e a seus respectivos testes de aceitação (especificados pelo cliente);
- Cada *User Story* é quebrada em atividades menores (caso exista necessidade);
- Cada atividade é alocada para um membro da equipe de desenvolvimento (a partir da Matriz de Competência);
- Cada membro da equipe de desenvolvimento estima o tempo de realização das atividades que foram alocadas para ele.

Quando se faz?

A alocação das *User Stories* deve ser feita apenas para o próximo *release*, pois planejar *releases* à frente significa que mudanças ou aparecimento de novos requisitos podem vir a invalidar um planejamento de atividades ainda distantes. O mesmo serve para o planejamento da iteração.

Duração: Cerca de 2 horas.

Obs.: Como o planejamento depende do cliente, e este nem sempre está acessível, o tempo dito acima tende a ser maximizado. Vale a pena investir tempo nesta fase, um bom planejamento reflete num ótimo andamento do projeto.

10. Implementação

Resumo

O que é?

É a realização das atividades estabelecidas no plano da iteração. Tem como principal artefato o código do sistema. Para que a implementação do sistema resulte em uma boa codificação é necessário que a equipe de desenvolvimento considere algumas práticas: **Integração Contínua**, **Boas Práticas de Codificação**, **Propriedade Coletiva de Código, Testes** e **Pequenos Releases**.

Para que serve?

Para que, a partir das atividades estabelecidas no plano da iteração, se obtenha uma versão codificada do sistema.

Quem faz?

Desenvolvimento e Testadores

Quando deve ser feito?

Durante todo o período de codificação, implementação.

Duração: O tempo de implementação está associado à duração da Iteração. Considerando que, o **YP** sugere 3 releases, cada um com duas iterações de 2 semanas cada, o tempo total necessário para a implementação são de, em média, 3 meses.

10.1. Integração Contínua

O YP propõe a integração contínua com o objetivo de melhorar o gerenciamento do projeto e o trabalho dos desenvolvedores, caso estes não possuam horários em comum. Para tanto é necessário que os módulos gerados do sistema sejam integrados continuamente, com o apoio de alguma ferramenta de controle de versão. Na verdade, é interessante que a integração seja feita cada vez que algum código novo seja implementado e esteja pelo menos sem erro de compilação para não comprometer o andamento do projeto.

No tocante ao gerenciamento do progresso do projeto pelo gerente, a integração contínua auxilia a coleta de métricas, tornando-a mais consistente, e conseqüentemente simplifica a geração do *Big Chart*, além de facilitar a análise real do desenvolvimento.

Com relação à equipe de desenvolvimento, o benefício é percebido, mais facilmente, em projetos cujos desenvolvedores não possuem horários de trabalhos em comum. Com a integração contínua a equipe não necessitará de um novo encontro para integrar tudo o que foi implementado separadamente, além de diminuir a quantidades de erros na implementação devido à integração de código.

10.2. Boas Práticas de Codificação

Uma vez que YP sugere a propriedade coletiva de código, deve-se ter em mente a necessidade da geração de um código limpo (evitar: linhas de código desnecessárias, repetição de trechos de código, implementação de funcionalidades não condizentes com as *User Stories*, tarefas não condizentes com o Modelo da Tarefa…) e de fácil entendimento. Com isso, um membro da equipe de desenvolvimento que não tenha codificado aquela parte do código será capaz de entendê-la e modificá-la, sem que seja necessário muito tempo ou esforço. Sugere-se aplicar ao projeto as práticas de *Design* Simples, Padrões de Codificação, Padrões de Projeto e Refatoramento, de forma que o código gerado possa ser considerado limpo. Segue uma breve descrição de cada uma dessas práticas.

Design **Simples**

Consiste em perseguir a geração do melhor código possível, ou seja, de fácil entendimento, sendo praticamente auto-explicativo, exigindo apenas comentários essenciais. Fazer testes com uma boa cobertura do código é imprescindível. O código deve ter um desempenho aceitável. Com relação à flexibilidade do código, deve-se ter cautela, uma vez que esta prática, quando em excesso, pode levar à produção de código desnecessário (linhas de código que não serão utilizadas durante a execução do sistema). A adição de código baseado na previsão de futuras funcionalidades deixa-o mais complexo. Seja sensato ao definir o nível de flexibilidade do código.

Padrões de Codificação

É bastante positivo que, antes de começar a codificar, a equipe de desenvolvimento defina um padrão de codificação. Deve-se determinar, por exemplo, o tamanho da identação utilizada, a forma como devem ser posicionados os parênteses e chaves, a maneira como devem ser nomeados os métodos e variáveis. Tudo isso para que os desenvolvedores não desperdicem tempo "corrigindo" a forma do código gerado por outro membro da equipe, e o entendimento seja o mesmo por toda a equipe.

Padrões de Projeto

O uso de soluções previamente pensadas por grandes projetistas da Orientação a Objetos fornece a possibilidade de reutilizar micro-arquiteturas de classes, objetos, suas funcionalidades e suas

colaborações. Ao utilizar essas soluções em diferentes tipos de problemas e situações, se ganha tempo, garantindo eficiência e fazendo com que o produto final aproxime-se de um software de qualidade. Mais detalhes sobre os Padrões de Projeto podem ser encontrados no livro da "*Gang of Four*" (GoF) [9] ou em [10].

Refatoramento

Pode ser visto como uma "pequena" modificação no código do sistema que não altera o seu comportamento funcional, mas que melhora algumas qualidades não-funcionais, tais como: simplicidade, flexibilidade, desempenho e clareza do código. Essa é uma prática bastante eficiente quando o objetivo é a limpeza de código. Porém, para que ocorra com sucesso, é necessário que todo o código esteja funcionando perfeitamente, ou seja, os testes de unidade devem cobrir todo o código, devem estar rodando 100%. Recomenda-se ter um controle de versões e condutas de integração contínua. É aconselhável fazer refatoramento constantemente, o que faz com que o código apresente-se cada vez mais legível. Para tanto, além das exigências anteriores, é preciso que o tempo para efetuá-lo seja estabelecido durante o planejamento da iteração. Existem diversos padrões de refatoramento [11].

10.3. Propriedade Coletiva de Código

Propriedade coletiva é a habilidade de qualquer membro da equipe poder alterar o código elaborado por outros membros durante a implementação. Isto implica dizer que o código é de posse coletiva, ou seja, toda a equipe é responsável por ele.

Esta prática é uma excelente ferramenta para a melhoria contínua de código, além de facilitar a recuperação de falhas, pois possíveis trechos de código problemáticos podem não ser claros para um dos desenvolvedores, mas podem ser facilmente percebidos pelos outros membros da equipe.

Para que estas idéias funcionem bem é imprescindível que haja um controle contínuo e eficiente de versões, para que se controle quem está alterando cada trecho de código em determinado momento, evitando conflitos. Outro pré-requisito fundamental é a certeza de se ter sempre o código testado de uma maneira eficaz, eficiente e funcional, de forma que, ao se alterar o código, a funcionalidade especificada permaneça a mesma. Outras necessidades importantes são a de se ter um código limpo e uma boa e consistente documentação, pois essas características facilitam o entendimento do código e diminuem o tempo gasto para compreensão deste quando é necessário efetuar alterações.

10.4. Testes

Testes têm por finalidade verificar se todos os requisitos do sistema forma implementados corretamente, ou seja, trata-se de uma análise das pré e pós-condições diante do contexto no qual está inserido o módulo a ser testado. A prática de efetuar bons testes assegura, na medida do possível, a qualidade e a corretude do sistema, além da satisfação do cliente e do usuário. Para tanto, é necessário testar o sistema sob diferentes aspectos (funcionalidade, usabilidade, carga, controle de acesso, desempenho, entre outros), o que classifica os teste em diferentes categorias, cada um com um objeto especifico de verificação. O YP recomenda que os **Testes de Unidade**, **de Aceitação** e **de Usabilidade** sejam feitos, não impedindo a realização dos demais, caso necessário**.**

O sucesso funcional do sistema está intimamente ligado à cobertura dos testes, nas suas unidades, nos módulos e no comportamento dos módulos submetidos aos testes de aceitação. Um módulo só deve ser considerado pronto depois de ser testado exaustivamente. Logo, os testes têm importância vital para o funcionamento do sistema. Uma *User Story* só é considerada implementada quando for submetida a todas as baterias de testes, passando com sucesso.

Teste de Unidade

Testam a estrutura interna do código, ou seja, a lógica e o fluxo de dados. Sendo assim, tais testes validam as menores partes do sistema (os métodos, classes ou trechos de códigos). Os testes de unidade devem ser escritos, preferencialmente, antes da construção do código [8], uma conduta que ajuda na solução do problema e melhora a qualidade do código desenvolvido. Tais testes devem ser elaborados e realizados pelos desenvolvedores, mas devem ser revisados pelos testadores. Os principais aspectos a serem validados com os testes de unidade estão relacionados à: manipulação de dados inconsistentes ou impróprios, condição de limite e tratamento de erros. Testes de unidade devem ser elaborados e codificados pelos desenvolvedores, antes ou durante a implementação do código.

Teste de Aceitação

Contemplam um conjunto de situações definidas pelo cliente sobre como medir o sucesso do projeto. Descrevem cenários que devem ser suportados pelo sistema, e estão sempre associados às *User Stories*. As

características destes testes devem ser extraídas do cliente da maneira mais transparente possível. A equipe deve deixar o cliente à vontade para definir como quiser os casos de testes mais críticos, além de apresentar sugestões que podem ser aprovadas por ele durante a conversa. Usuários finais podem ser ouvidos para reforçar aspectos levantados pelo cliente. Em suma, os testes de aceitação são elaborados pelos clientes, durante o levantamento das *User Stories*, mas são gerados (realizados) pelos testadores quando a *User Story* associada ao mesmo é implementada.

Teste de Usabilidade

Baseia-se em uma combinação de um conjunto de técnicas que incluem observação, questionários, entrevistas e testes com o usuário. Os testes de usabilidade verificam se os objetivos de usabilidade especificados pelo cliente e pelo usuário foram satisfeitos, além de averiguar como está sendo o processo de interação entre o usuário e o sistema. Os cenários de teste de usabilidade devem buscar avaliar o desempenho de usuários típicos na realização de tarefas comuns.

O YP sugere que, alguns usuários sejam recrutados, de acordo com o perfil definido pelo cliente, a fim de utilizar o sistema para realizar um conjunto de atividades a partir de um roteiro (roteiro de atividades).

Durante a execução das atividades, a equipe de desenvolvimento deve ficar atenta a fim de identificar falhas da interação entre o usuário e a interface do sistema, que são evidenciados através de erros cometidos, buscas pela ajuda, não realização das atividades estabelecidas, entre outros. O tempo de realização das atividades também pode ser um indicador interessante, pois, se o usuário (de teste) leva mais tempo do que o previsto para realizar uma atividade, é possível que o problema esteja relacionado à maneira como o sistema está se apresentando, ou seja, com a interface.

Após a execução do roteiro de atividades o usuário deve responder a um questionário pós-teste, onde o mesmo irá indicar o seu grau de satisfação ao utilizar o produto. Diante do *feedback* obtido com os testes de usabilidade a equipe de desenvolvimento deve sanar os problemas detectados.

Os testes de usabilidade são elaborados pelos testadores (o modelo da tarefa pode ser útil), e realizados pelos usuários quando se tem uma versão completa do sistema. O procedimento necessário para a realização do teste de usabilidade pode ser visto, sucintamente, abaixo:

- Elaborar um roteiro de atividades;
- Recrutar usuários de teste;

- Observar o usuário executando as atividades do roteiro (se possível em um ambiente monitorado - laboratório de usabilidade);
- Colher indicadores quantitativos para análise (número de ações incorretas, consultas a ajuda, erros, repetidos, atividades concluídas...);
- Aplicar questionário pós-teste (para Sondagem da Satisfação Subjetiva do Usuário);
- Analisar os resultados obtidos e sanar as falhas detectadas.

Dica

Uma prática, antes presente em condutas de inspeção, que prima pela qualidade e capacidade do código, é a Revisão de Código. Tal prática auxilia na melhoria da qualidade e da funcionalidade do código através de revisões internas. Tais revisões são realizadas pelos testadores. Cabe ao gerente determinar o momento mais adequado para a equipe efetuar a revisão de código. É importante não acumular código para revisão, pois quando isso ocorre, o tempo de revisão pode ser maior que o desejado. Recomenda-se que cada testador revise o código gerado por um outro membro da equipe, reforçando a idéia de propriedade coletiva de código.

A necessidade de revisões sucessivas de código pode ser minimizada através da prática de programação em pares [10], na qual a produção de código é realizada por duas pessoas, lado a lado numa mesma máquina. Uma pessoa programa enquanto a outra pensa se aquela é a melhor maneira de se construir aquele código. Estas duplas devem se alternar após um certo período, geralmente a cada dia, por exemplo. O gerente pode decidir quando e como melhor alternar as duplas, a fim de se obter um melhor desempenho da equipe. Observe que as condutas de revisão de código e programação em pares podem ocorrer juntas, porém se a equipe não consegue dispor de horários compatíveis entre os programadores, as revisões de código podem ser planejadas junto com os testes.

10.5. Pequenos Releases

Normalmente, no desenvolvimento de software, o prazo para finalização do sistema é uma preocupação crítica. Tratando-se do YP, é necessário que o planejamento esteja sempre visando o cumprimento dos requisitos alocados em cada *release*.

O YP propõe que o planejamento seja focado em pequenos *releases,* garantindo uma maior interação com o cliente. As *User Stories* alocadas em cada *release* só podem ser consideradas como finalizadas após a realização dos testes de aceitação. Um benefício desta prática é a redução do impacto das mudanças de requisitos.

Considerando que cada *User Story* alocada seja finalizada após a aceitação do cliente, pode-se considerar que tudo o que esteja implementado funcione satisfazendo a especificação do projeto. Mesmo que haja erros na implementação de algum requisito, o impacto é bem menor do que realizar uma implementação com *release* mais longo.

Um período letivo nos moldes atuais possui quatro meses de duração, o ideal é que cada *release* dure cerca de um mês, contendo duas iterações, cada qual com duas semanas de duração. A cada finalização de iteração sugere-se um novo encontro com o cliente do projeto.

Pequenos *releases* resultam em:

- Melhorias na interação com o cliente do projeto;
- Minimização do impacto sobre os erros na implementação;
- Redução do impacto nas mudanças de requisitos observados pelo cliente;
- Melhorias na identificação e superação dos riscos.

11. Reunião De Acompanhamento

A reunião de acompanhamento deve ocorrer **semanalmente**, visando avaliar sistematicamente os resultados obtidos no projeto até o momento. A mesma deve ser coordenada pelo gerente de projeto, que tem fundamental importância na análise dos resultados obtidos. A dedicação e o compromisso do gerente é uma questão decisiva para o sucesso do projeto. Seu empenho e atenção podem identificar, prematuramente, falhas no andamento do projeto. Quando a descoberta de deficiências é feita antecipadamente, é possível minimizar os custos tanto para a equipe de desenvolvimento quanto para o cliente.

Toda a equipe de desenvolvimento deve participar da reunião, pois as discussões se darão em torno da produção de seus membros, assim como das situações vivenciadas por cada um deles durante a semana. Para facilitar a avaliação, é rigorosamente recomendado fazer uso do *Big Chart* e da Tabela de Alocação de Atividades (TAA). Uma análise de riscos e possíveis mudanças inerentes ao projeto devem ser levantadas durante a reunião, assim como a necessidade explícita de refatoramento de código.

Big Chart

Pode ser visto como uma análise quantitativa do andamento do projeto a partir do recolhimento de métricas. As métricas são definidas pelo gerente de projeto, de acordo com os pontos principais a serem analisados. Sendo assim, podem ser recolhidos e avaliados: o número de classes existentes, o número de linhas de código, os testes de aceitação que estão prontos, os testes de unidade que estão rodando, os módulos do sistema, dentre outros. Todos esses dados são exibidos, em forma de tabela ou gráfico.

Ao analisar o *Big Chart*, não se deve esquecer que o YP trabalha com *releases* e iterações de tempo fixo, quatro semanas e duas semanas, respectivamente. Esses são dados fundamentais para a análise. Caso o *Big Chart* não revele mudanças entre uma semana e outra, ou mostre algo inesperado ou mesmo estranho (o aumento do número de classes geradas e a permanência do número de classes de testes existentes, por exemplo), o gerente deve chamar a atenção da equipe, procurando respostas para a "anormalidade" ali presente.

O *Big Chart* pode ser considerado um dos instrumentos mais importantes para auxiliar o gerente na análise de resultados. Quando o gerente de projeto possui a percepção aguçada diante das relações

existentes entre os dados expostos no *Big Chart*, seu trabalho junto à equipe de desenvolvimento e ao cliente é qualitativamente satisfatório. O auxílio de ferramentas para coleta de métricas e geração automática do *Big Chart* pode ser bastante útil.

Tabela de Alocação de Atividades (TAA)

Quando a reunião coincide com o término de uma iteração, a TAA deve ter seu preenchimento finalizado. Só ao final da iteração é possível preencher os campos de tempo real necessário para realizar cada uma das atividades e o *status* referente às mesmas. A TAA começa a ser preenchida no planejamento da iteração.

Refatoramento

Necessidade explícita de refatoramento é identificada pelo desenvolvedor, quando o mesmo está ciente de que o código que gerou precisa passar por uma revisão de código ou mesmo refatoramento. Essa necessidade não deve ocorrer com freqüência, pois os programadores devem tentar produzir o seu melhor código.

Análise de riscos

Riscos caracterizam situações indesejáveis que podem ocorrer durante o processo, impedindo o fluxo normal do desenvolvimento do software. Quando riscos são identificados e analisados, deve-se buscar uma solução prudente para que o projeto não venha a falhar. De acordo com o tipo de risco encontrado, novas diretrizes para o projeto devem ser traçadas. Caso essa providência não seja suficiente, o gerente, junto aos desenvolvedores, deve questionar a viabilidade do projeto, podendo este ser abortado após o entendimento e consentimento do cliente.

A análise de riscos busca, entre outras coisas, avaliar o projeto rotineiramente e aumentar a probabilidade de sucesso do mesmo, a partir da superação dos riscos. Nessa análise, é importante que sejam propostos meios de se resolver os problemas encontrados. É necessário que exista uma metodologia de análise de riscos bem definida durante o processo de desenvolvimento, no caso do YP, durante as reuniões de acompanhamento.

Uma lista de riscos deve ser construída com a descrição do risco, a data em que o risco foi identificado, a prioridade de superação (classificadas nos níveis Alta, Média e Baixa), o responsável (não é necessariamente quem identificou/provocou o risco, mas sim quem irá buscar formas de superar o risco), e a solução encontrada, quando o risco

tiver sido superado. Essa memória é particularmente importante quando novos problemas de mesma natureza aparecerem no processo, pois ficará mais fácil eliminá-los a partir da solução apresentada. A partir dessa tabela de riscos, pode-se identificar quais são os maiores empecilhos encontrados durante o desenvolvimento do projeto. Atacam-se primeiro os riscos de maior prioridade e os que estão a mais tempo no processo. A relação entre prioridade e tempo deve ser a maneira mais natural de se priorizar os riscos.

A existência de uma metodologia de análise de riscos e uma rotina gerencial que privilegie tal prática faz com que os desenvolvedores estejam livres de surpresas indesejáveis quanto ao resultado final do projeto. Além da análise sistemática dos riscos, técnicas como o uso de padrões de projeto e refatoramento têm o intuito de diminuir os riscos de fracasso em projetos de software.

Mudança

A identificação de possíveis mudanças deve ser abordada durante a reunião. Caso seja encontrada alguma mudança, esta deve ser tratada com bastante cuidado. Se a mesma reflete em artefatos já gerados, a atualização dos mesmos deve ser feita sob orientação do gerente, o qual deve ser cauteloso para que os artefatos continuem consistentes após as alterações.

Existem vários níveis de mudanças: aquelas que exigem apenas modificações de implementação, ou alocação de tarefas; e aquelas que podem mudar o planejamento, o projeto arquitetural, o modelo lógico de dados, a priorização das *User Stories*, a interface e até mesmo o documento de visão. Cada uma delas deve ser tratada com atenção para que a melhor solução seja obtida.

A participação efetiva do gerente é fundamental nessa decisão. Quando as mudanças trazem riscos ou mudanças de requisitos, é possível que a equipe não consiga cumprir o que tinha ficado firmado com o cliente. É aconselhável variar o escopo do prometido e não o tempo necessário para que o mesmo seja efetuado, entretanto a decisão de alteração do escopo deve ser discutida com o cliente antecipadamente.

Caso a modificação ocorra no escopo de tempo, é possível que a equipe perca a credibilidade junto ao cliente, pois este quer ver resultados concretos do projeto, mesmo que em menor quantidade do que a presente no acordo. Não devemos esquecer que o YP trabalha com *releases* e iterações de tempo fixo, ou seja, o tempo planejado para cada *realease* e iteração não deve ser modificado, apenas o escopo do projeto pode ser ajustado (aumentado ou diminuído).

Resumo

O que é?

Reunião semanal que visa recolher e analisar métricas. É aqui que se faz uso do *Big Chart* e da Tabela de Alocação de Atividades (TAA) para examinar o andamento do projeto. A preocupação com os riscos e possíveis mudanças deve ser pauta da reunião, assim como a necessidade explícita de refatoramento.

Para que serve?

Para fornecer ao gerente uma visão dos resultados obtidos pela equipe em uma semana de trabalho. A partir da análise feita, o gerente é capaz de identificar os pontos nos quais o projeto está caminhando bem, e aqueles onde existem falhas, podendo assim, buscar soluções para melhorar os resultados futuramente obtidos.

Quem faz?

A reunião deve ser coordenada pelo gerente de projeto. Todos os desenvolvedores de testadores devem comparecer. A presença de todos é extremamente importante. A dedicação do gerente é decisiva com relação ao sucesso do projeto.

Duração: Cerca de 1 hora e 30 min.

12. Exemplos

Esta sessão tem por objetivo exemplificar todos os artefatos que devem ser gerados ao longo do fluxo do easYProcess.

Todos os exemplos estão dentro de um mesmo contexto, trata-se de um sistema que apóia agentes de saúde do município a documentar, digitalmente, os registros necessários para o funcionamento do PSF (Programa de Saúde na Família), tanto no que diz respeito as famílias favorecidas pelo PSF, quanto aos seus membros (equipe de médicos, enfermeiros, dentistas...) e ao fornecimento de medicamentos a cada unidade do PSF, em uma determinada região.

Os artefatos aqui presentes são fruto da implantação real do YP por uma equipe de alunos ao desenvolver o PSFcomponent na disciplina de LES [7].

12.1. Exemplo Definição de Papéis

Quadro 1- Definição de papéis

Equipe	Papéis
Francilene Garcia	Cliente
Luiza Maria	Usuário
Ana Esther	Gerente, Desenvolvedor, Testador
Helton Lima	Gerente, Desenvolvedor, Testador
Lucas Albertins	Gerente, Desenvolvedor, Testador

Obs.: Uma tabela como esta deve ser produzida a cada *release*, refletindo o rodízio de papéis dentro da equipe. Cliente e Usuários possivelmente não irão mudar.

12.2. Documento de Visão

Este artefato é composto por 5 sub-partes, a saber: (i) descrição do sistema a ser desenvolvido; (ii) requisitos funcionais; (iii) requisitos não funcionais; (iv) perfil do usuário; e, por fim, (v) objetivos de usabilidade.

12.2.1. Descrição do Sistema

O Ministério da Saúde criou, em 1994, o Programa Saúde da Família (PSF). Este programa visa reorganizar a prática de atenção à saúde, colocando-a em novas bases, substituindo o modelo tradicional, com a finalidade da melhoria da qualidade de vida da população. Na Paraíba, o PSF atua em várias cidades, englobando um número significativo de famílias. O atendimento das famílias é realizado através de postos de atendimento, que são agrupados em torno de uma Secretaria de Saúde do município. Cada posto de atendimento reúne informações sobre as famílias. Cada unidade PSF atende entre 600 e 1500 famílias. Essas informações podem compartilhadas entre os postos e enviadas à Secretaria. As famílias precisam estar cadastradas na Secretaria de Saúde do município para poderem utilizar-se do atendimento provido pelas unidades PSF. Entretanto, toda essa coleta de informação é feita manualmente, sem a utilização de qualquer informatização específica para o programa.

Podemos ver que há uma grande necessidade de um sistema de informação sobre esses habitantes, que pode ser iniciado com a construção de um sistema de cadastramento informatizado. Como o cadastro é feito manualmente, há uma grande dificuldade no mapeamento, gerenciamento e busca de várias informações sobre a população atendida.

Outra deficiência do programa é a questão dos medicamentos das farmácias cadastradas. Cada farmácia recebe uma quantidade de medicamentos e distribui para as unidades PSF. É necessário um sistema que controle a quantidade e a descrição dos medicamentos que passam pelas farmácias.

Neste caso, temos dois componentes para desenvolver, o componente de cadastramento do PSF (famílias favorecida, região contempladas, equipe de profissionais responsáveis...) e o componente de controle de medicamentos das unidades PSF. Iremos verificar a viabilidade do desenvolvimento de ambos durante a fase de planejamento.

Inicialmente, iremos dar prioridade para a construção do primeiro componente do **PSF*component***.

12.2.2. Requisitos Funcionais

Componente 01 - Cadastramento do PSF

- Cadastro de unidades PSF;
- Cadastramento de pessoas;
- Busca de informações em relação a: bairros tendidos, ruas, comunidades, famílias, CEP...
- Geração de relatórios.

Componente 02 - Controle de Medicamento das Unidades PSF

- Cadastro de unidades PSF;
- Cadastro de remédios;
- Adicionar remédios ao estoque;
- Enviar remédios a uma unidade PSF;
- Busca de informações sobre os remédios: quantidade recebida, quantidade repassada a uma determinada unidade PSF, quantidade no estoque.

12.2.3. Requisitos Não Funcionais

- Interface Web (JSP + HTML);
- Segurança - Haverá necessidade de *login* para utilizar os sistemas;
- Banco de dados MySQL;
- Arquitetura N camadas;
- Utilização dos padrões de projeto (VO e DAO);
- Material de treinamento.

12.2.4. Perfil do Usuário

Agente da secretaria de saúde do município, com idade média de 32 anos de idade. Em sua maioria, possui pouco conhecimento de informática. No entanto, possui alta capacidade de aprendizado e um nível médio de curiosidade com relação ao uso de softwares. Desta maneira o treinamento é necessário, e suficiente, para que o usuário se familiarize com o sistema.

12.2.5. Objetivos de Usabilidade

Quadro 2 - Objetivos de Usabilidade

Objetivo	Mensuração
Reduzir a taxa de erros	Número de tarefas concluídas sem falhas.
Facilitar o aprendizado	Uso de recursos avançados.
Adequar conteúdo (terminologia e simbologia)	Aprendizado mantido mesmo com uso pouco freqüente do sistema.
Aumentar a satisfação subjetiva do usuário	Confrontar a primeira impressão do usuário com sua opinião após uso prolongado do sistema.
Manter clareza na estrutura	Número de ações incorretas, de erros repetidos e consulta à ajuda (*online* ou *offline*).
Ser atrativo ao usuário	Sondar a satisfação subjetiva do usuário ao utilizar o sistema.
Possuir telas simples	Observar dificuldades de navegação.
Manter consistência na seqüências de ações necessárias para realizar uma tarefa	Número de opções incorretas e de erros repetidos
Disponibilizar boa documentação	Facilidade de encontrar as informações desejadas ao utilizar os manuais de ajuda sejam estes *online* ou *offline*

12.3. Modelo da Tarefa
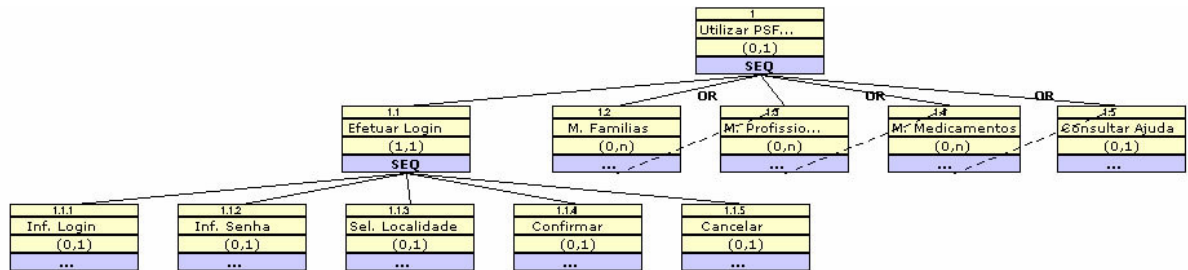
Figura 4 – Modelo da Tarefa: Utilizar Sistema PSFcomponent
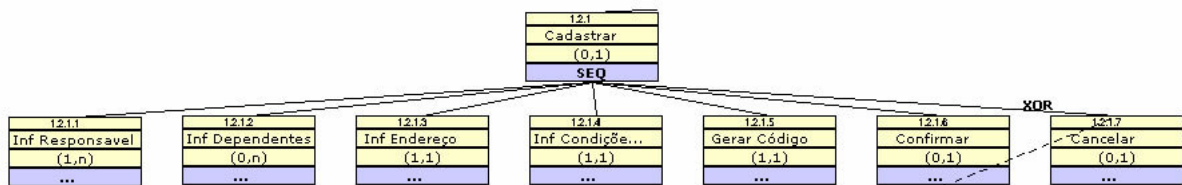
Figura 5 - Modelo da Tarefa: Cadastrar Familias/Membros
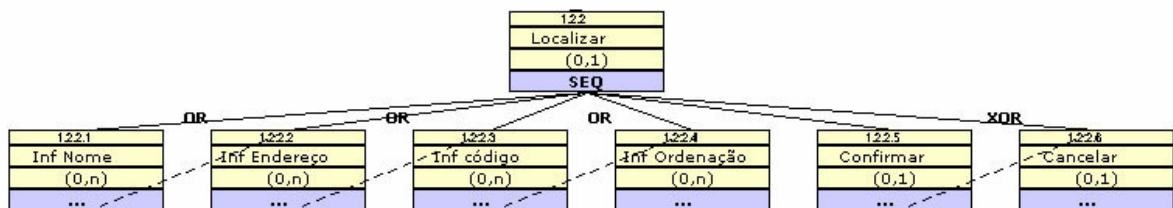
Figura 6 - Modelo da Tarefa: Localizar Familias/Membros
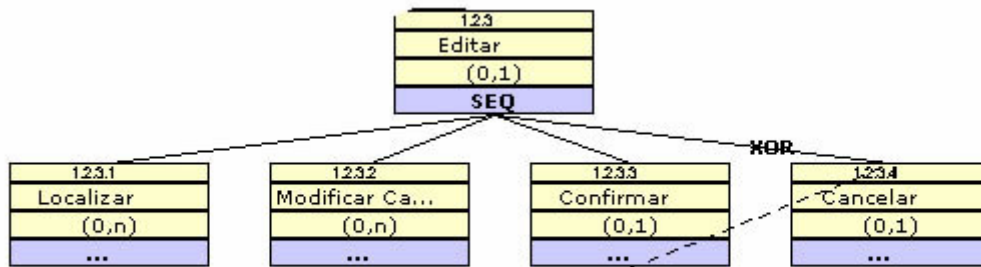
Figura 7 - Modelo da Tarefa: Editar Familias/Membros
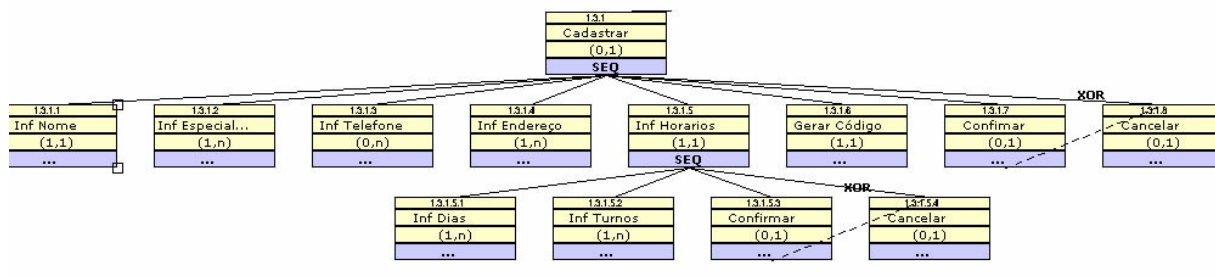
Figura 8 - Modelo da Tarefa: Cadastrar Profissionais (Membros da Equipe do PSF)
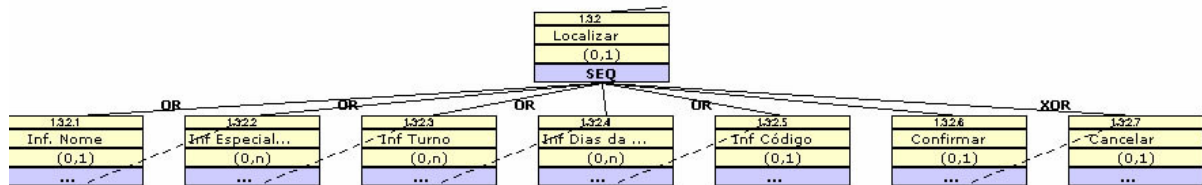
Figura 9 - Modelo da Tarefa: Localizar Profissionais (Membros da Equipe do PSF)
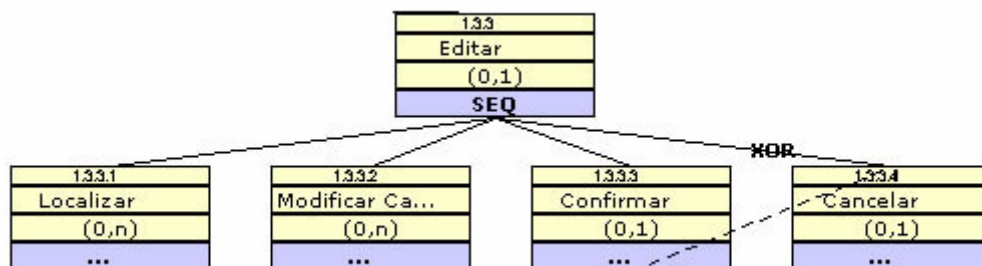
Figura 10 - Modelo da Tarefa: Editar Profissionais (Membros da Equipe do PSF)
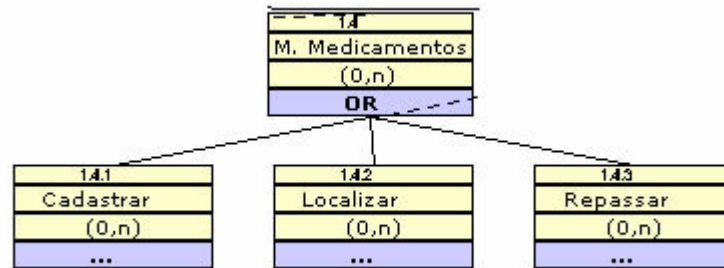
Figura 11 - Modelo da Tarefa: Manipular Medicamentos
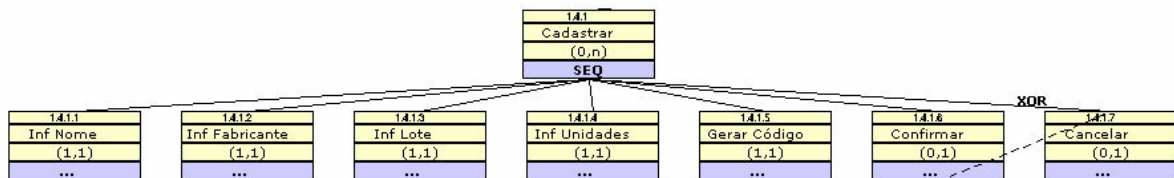
Figura 12 - Modelo da Tarefa: Cadastrar Medicamentos
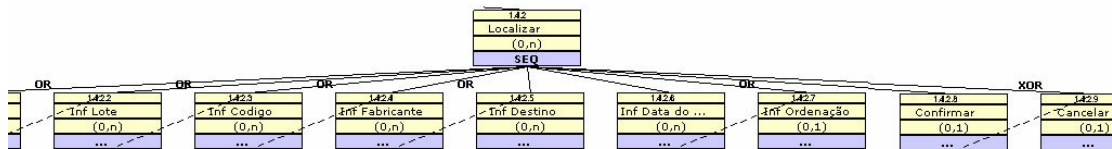
Figura 13 - Modelo da Tarefa: Localizar Medicamentos
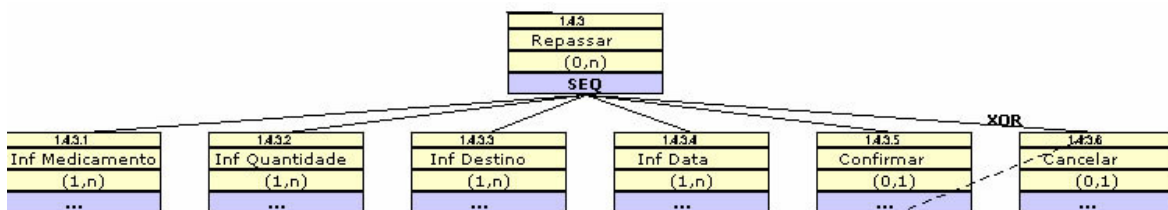
Figura 14 - Modelo da Tarefa: Repassar Medicamento para as Unidades do PSF

12.4. *User Stories* e Testes de Aceitação

Quadro 3 - Definição das User Stories e seus Respectivos Testes de Aceitação

US01	Estudar JSP, BD MySQL, DAO, VO e mecanismos de testes a serem utilizados. Gerar exemplos como resultados. **Estimativa inicial:** 12h
TA1.1	Verificar de os exemplos gerados satisfazem os clientes a fim de confirmar o uso das tecnologias citadas.
US02	Implementar funcionalidade de cadastro de famílias **Estimativa inicial:** 11h
TA2.1	Cadastrar uma família com todos os seus dados corretos (Cadastro Efetuado com Sucesso)
TA2.2	Cadastrar uma família sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)
TA2.3	Cadastrar uma pessoa com todos os seus dados corretos (Cadastro Efetuado com Sucesso)
TA2.4	Cadastrar uma pessoa sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)
US03	Implementar funcionalidade de buscas diversas **Estimativa inicial:** 13h
TA3.1	Recuperar dados gerais da família a partir de uma chave existente (Dados devem ser retornados com sucesso)
TA3.2	Recuperar dados gerais da família a partir de uma chave inexistente (Mensagem de erro deve ser retornada)
TA3.3	Recuperar dados referentes a moradia da família a partir de uma chave existente (Dados devem ser retornados com sucesso)
TA3.4	Recuperar dados referentes a moradia da família a partir de uma chave inexistente (Mensagem de erro deve ser retornada)
TA3.5	Recuperar dados sobre os membros da família passando uma chave

	existente (Dados devem ser retornados com sucesso)
TA3.6	Recuperar dados sobre os membros da família passando uma chave inexistente (Mensagem de erro deve ser retornada)
US04	Implementar funcionalidade de cadastro de unidades PSF **Estimativa inicial:** 6h
TA4.1	Cadastrar uma unidade de PSF informando todos os campos obrigatórios (Cadastro Efetuado com Sucesso)
TA4.2	Cadastrar uma unidade de PSF sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)
US05	Implementar funcionalidade de remoção e alteração de dados **Estimativa inicial:** 9h
TA5.1	Alterar endereço da familia a partir da chave específica
TA5.2	Fazer busca pelo endereço da familia a partir da mesma chave especificada a cima (O novo endereço deve ser exibido)
TA5.3	Excluir a familia X, a partir de sua chave, por motivo de mudança de endereço
TA5.4	Fazer busca pela chave referente a família X (Nenhum registro deve ser exibido)
US06	Incluir funionalidade de autenticação (restrição de acesso) **Estimativa inicial:** 6h
TA6.1	Acessar sistema a partir de um login válido (autenticação feita com sucesso)
TA6.2	Acessar sistema a partir de um login inválido (mensagem de erro deve ser exibida)
US07	Elaborar manual **Estimativa inicial:** 6h
TA7.1	Procurar por informações diversas no manual

12.5. Protótipo da Interface
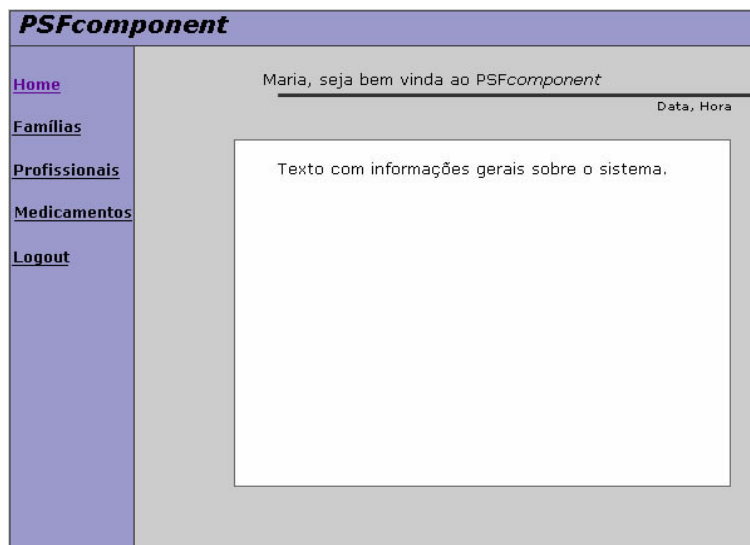
Figura 15 - Protótipo da Interface: Acesso ao Sistema PSFcomponent

Figura 16 - Protótipo da Interface: Tela Inicial após o Login

Figura 17 - Protótipo da Interface: Tela de Cadastro (Família/Membro)

12.6. Projeto Arquitetural
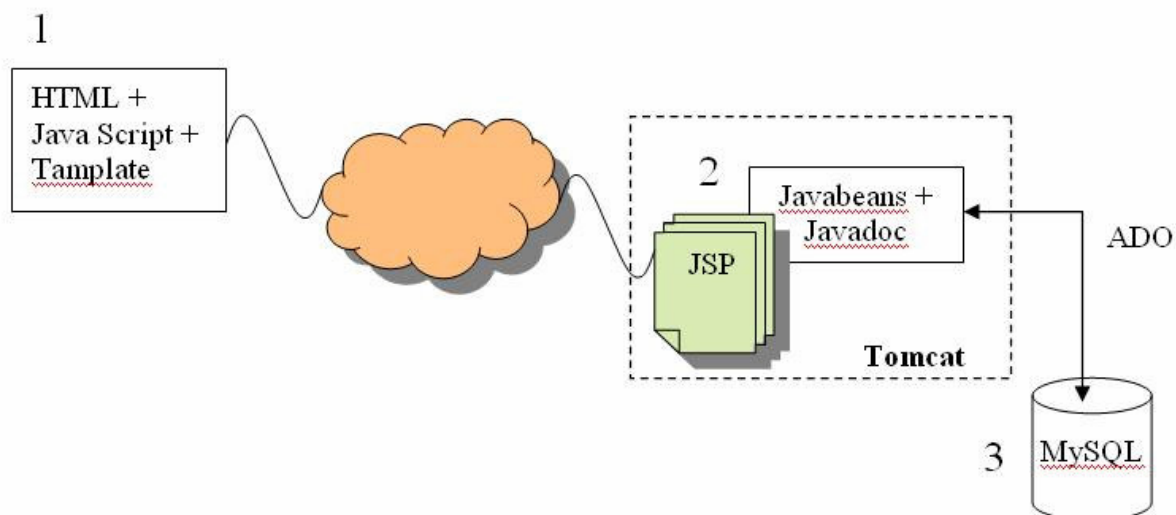
Figura 18 - Projeto Arquitetural do Sistema PSFcomponent

Descrição do Projetos Arquitetural

A arquitetura para componentes verticais, que será usada nos dois primeiros componentes já em fase de construção em CG, segue o padrão n-camadas, separando a apresentação (ou visão), as camadas de negócios e a camada de persistência de dados. De acordo com o padrão definido, as camadas serão implementadas em tecnologias *web*, abertas e *Open Source*.

Na interface serão utilizadas páginas HTML, com inserção de *scripts* JavaScript para operações de validação de entradas. A apresentação destas páginas obedecerá padrões de interface e navegação definidos em um *tamplate* a ser apresentado pela equipe FLO-PREF.CG.

Embora se proponham inicialmente a serem executadas completamente em uma máquina, que estará *standalone*, a arquitetura utiliza tecnologia *web*, sendo portanto aplicável em ambientes de intranet/Internet. A seguir, estão descritas as características do serviço de hospedagem de páginas:

- Servidor *web* + servidor java *pages*: *Tomcat*
- Páginas JSP: as requisições do *browser* se destinarão a páginas JSP, que obedecerão a algumas restrições de conteúdo.

- Nenhum algoritmo, regra de negócio ou acesso a dados será implementado em código dentro das páginas JSP, entre *tags* de *script*. Toda lógica da aplicação, bem como acesso a dados, deve ser implementada em componentes JavaBeans, a serem referenciados nas páginas JSP;

- A persistência dos objetos deve ser implementada sob o padrão de projeto DAO: http://java.sun.com/blueprints/patterns/DAO.html;

- A comunicação entre as camadas deve ser implementada sob o padrão de projeto VO: http://www.mundooo.com.br/php/modules.php?name=News&file=print&sid=483;

- Os componentes JavaBeans serão acompanhado de documentação no padrão JavaDoc, que deve ser produzida obedecendo ao *template* a ser disponibilizado pela equipe FLOPREF.CG.

Será adotado como padrão o SGBD relacional MySQL, por suportar os requisitos de persistências existentes em aplicações da complexidade abordada no escopo do projeto FLO-PREF. Trata-se de um Software Livre, disponível para *download* em http://dev.mysql.com/downloads/mysql/5.0.html.

12.7. Modelo Lógico de Dados

Cadastramento do PSF
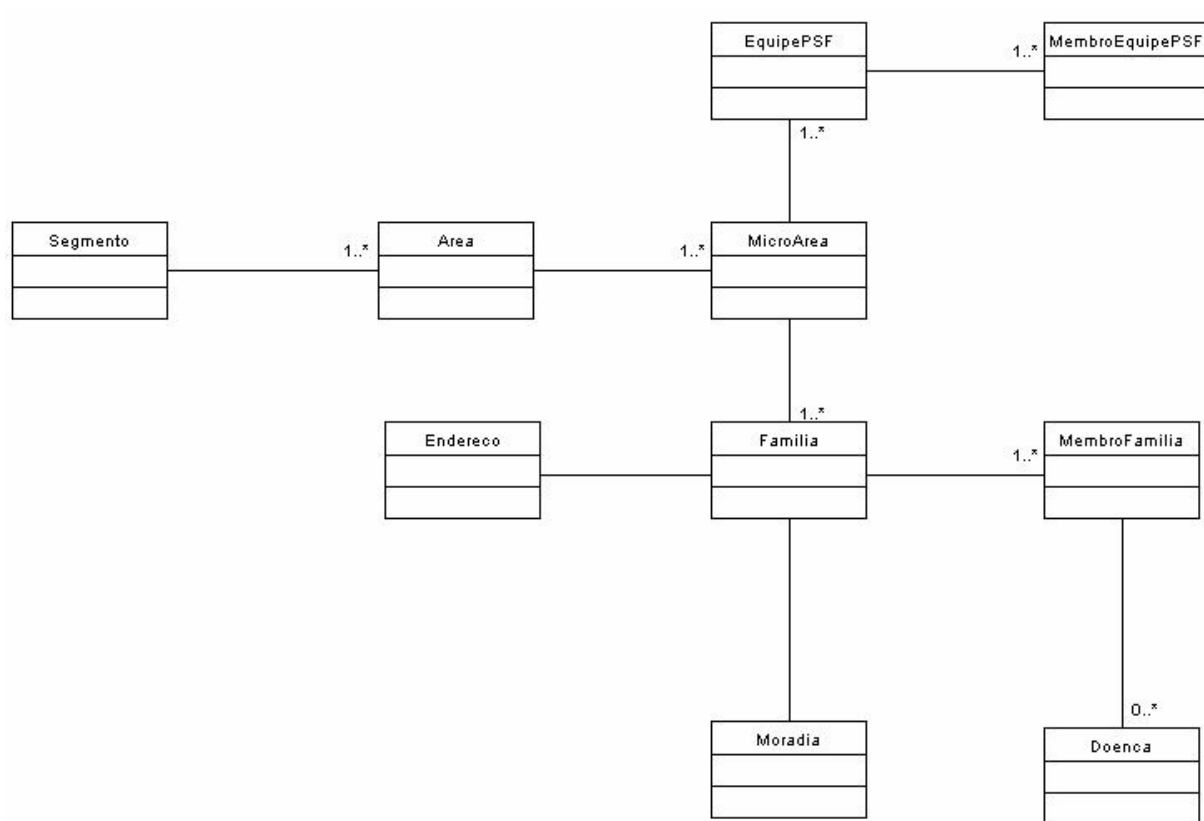
Figura 19 - Modelo Lógico de Dados: Componente 01

Controle de Medicamento das Unidades PSF
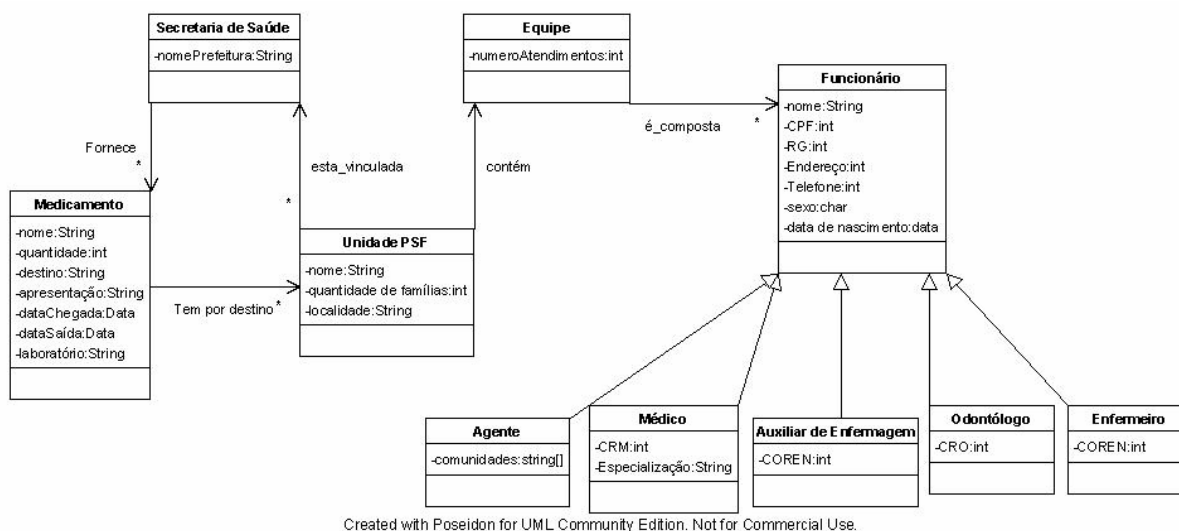
Figura 20 - Modelo Lógico de Dados: Componente 01

12.8. Matriz de Competências

Quadro 4 - Matriz de Competências

Equipe	Competências
Ana Esther	JSP - HTML - JAVA - TAOS
Helton Lima	JSP - HTML - JAVA - SWT
Lucas Albertins	JSP - HTML - JAVA - MySQL

Obs.: Ao final do semestre letivo é interessante que seja gerada uma nova Matriz de Competências para que, comparando-a com a matriz inicial, possa-se saber quais competências foram adquiridas ao longo do desenvolvimento do projeto.

12.9. Plano de Release

Quadro 5 - Plano de Release 01

Release 01: 13/09 – 27/09	**Gerente - Helton**	
Iteração	***User Story***	**Período**
Iteração 01	US01	13/09 – 20/09
Iteração 02	US02	21/09 – 27/09

Quadro 6 - Plano de Release 02

Release 02: 28/09 – 21/10	**Gerente - Lucas**	
Iteração	***User Story***	**Período**
Iteração 03	US03	28/09 – 05/10
Iteração 04	US04	05/10 – 21/10

Quadro 7 - Plano de Release 03

Release 03: 22/10 – 04/11	**Gerente – Ana Esther**	
Iteração	***User Story***	**Período**
Iteração 05	US05	22/10 – 28/10
Iteração 06	US06, US07	29/10 – 04/11

12.10. Plano de Iteração – Início da Iteração

Quadro 8 - Plano de Iteração: Tabela de Alocação de Atividades do Início da Interação

Iteração 02 - (21/09/05 – 27/09/05)		
US02 - Implementar funcionalidade de cadastro de famílias		
Testes de Aceitação		***Status***
TA2.1	Cadastrar uma família com todos os seus dados corretos (Cadastro Efetuado com Sucesso)	
TA2.2	Cadastrar uma família sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	
TA2.3	Cadastrar uma pessoa com todos os seus dados corretos (Cadastro Efetuado com Sucesso)	
TA2.4	Cadastrar uma pessoa sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	

Atividade	**Descrição**	**Responsável**	**Estimativa de tempo**	**Tempo real**	***Status***
A2.1	Implementar interface para cadastro das famílias	Helton	4		
A2.2	Gerar *script* para configuração do MySQL	Lucas	3		
A2.3	Implementar funcionalidade para cadastro das famílias	Ana Esther e Lucas	4		
A2.4	Instalar aplicação em servidor	Helton	1		
A2.5	Gerar *script* dos TA	Ana Esther	1		

Observações:

- A TAA só é totalmente preenchida durante a reunião de acompanhamento, ao final de cada Iteração. Só então as infomações sobre tempo real e *status* estão disponíveis;
- A estimativa de tempo feita durante a definição das *User Stories* nem sempre são mantidas quando a *User Story* é decomposta em atividades menores (As atividades da *User Story* 02, somam 13 horas, diferente da estimativa inicial de 11 horas);
- A notação TA2.3 significa: Teste de Aceitação 3 para a *User Story* 2;
- A notação A2.1 significa: Atividade 1 da *User Story* 2;
- Os *status* C – concluído; D – em desenvolvimento A – abortada podem ser representados por imagem (, e , respectivamente), caso esta representação seja mais sigificativa para a equipe de desenvolvimento.

12.11. Plano de Iteração: Ao final da iteração

Quadro 9 - Plano de Iteração: Tabela de Alocação de Atividades do Fim da Interação

Iteração 02 - (21/09/05 – 27/09/05)		
US02 - Implementar funcionalidade de cadastro de famílias		
Testes de Aceitação		***Status***
TA2.1	Cadastrar uma família com todos os seus dados corretos (Cadastro Efetuado com Sucesso)	●
TA2.2	Cadastrar uma família sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	●
TA2.3	Cadastrar uma pessoa com todos os seus dados corretos (Cadastro Efetuado com Sucesso)	●
TA2.4	Cadastrar uma pessoa sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	●

Atividade	**Descrição**	**Responsável**	**Estimativa de tempo (horas)**	**Tempo real (horas)**	***Status***
A2.1	Implementar interface para cadastro das famílias	Helton	4	4	●
A2.2	Gerar *script* para configuração do MySQL	Lucas	3	5	●
A2.3	Implementar funcionalidade para cadastro das famílias	Ana Esther e Lucas	4	3	●
A2.4	Instalar aplicação em servidor	Helton	1	1,5	●
A2.5	Gerar *script* dos TA	Ana Esther	1	2	●

Observações:

- A TAA está totalmente preenchida;
- **As atividades não concluídas devem ser realocadas na iteração seguinte**;
- **A US só é considerada concluída se todos os testes de aceitação forem executados com sucesso e se todas as suas atividades forem concluídas;**
- A estimativa de tempo feita para cada atividade pode não ser real, a atividade A2.2 por exemplo, levou mais tempo do que o esperado (A medida que o projeto avança, as estimativas iniciais ficam mais próximas do tempo real de realização das atividades);
- Pode-se acrescentar, ao final da TAA, um espaço para comentários gerais, com o propósito, por exemplo, de justificar a não conclusão de uma atividade;
- Os *status* C – concluído; D – em desenvolvimento A – abortada podem ser representados por imagem (, e , respectivamente), caso esta representação seja mais sigificativa para a equipe de desenvolvimento.

12.12. *Big Chart*

Quadro 10 - Big Chart

Acompanhamento da Coleta de Métricas							
Data	**Classes**	***Scripts***	**Testes de Aceitação**	**Testes de Unidade**	**Páginas ***	***User Stories***	**Observações**
13/09	0	0	0	0	0	0	Não houve reuso de código ou páginas JSP/HTML.
20/09	12	3	1	1	6	1	-
04/10	16	4	4	10	10	2	-
21/10	15	6	11	8	15	3	Classe removida junto com respectivos testes
28/10	19	9	13	19	26	5	-
04/11	28	8	20	43	30	7	-

* HTML/JSP-Servlet
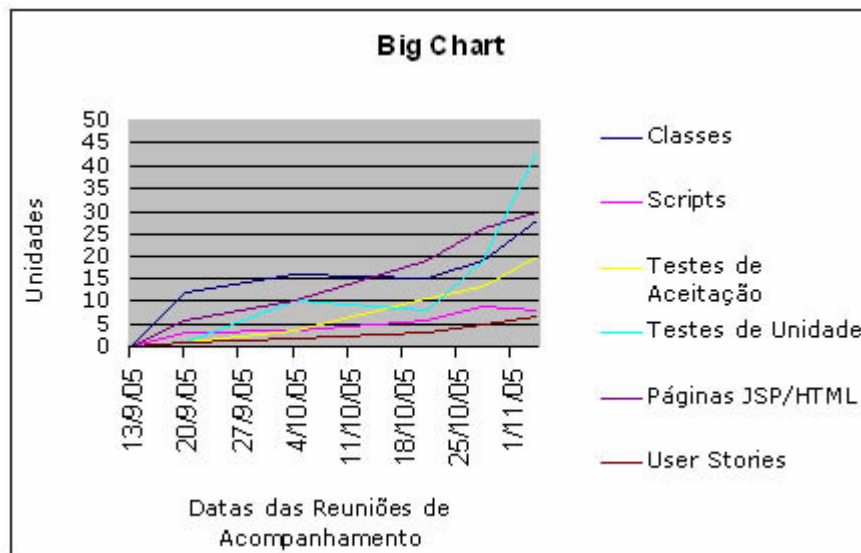
Figura 21 - Big Chart

Observações:

- O *Big Chart* deve ser atualizado a cada reunião de acompanhamento (final de cada Iteração).

12.13. Análise de Riscos

Quadro 11 - Quadro para Análise de Riscos

Data	Risco	Prioridade	Responsável	Status	Providência/Solução
13/09	Uso da tecnologia desconhecida (DAO, VO)	Alta	Todos	**Superado**	Estudar o assunto, considerando o tempo necessário no plano de iteração.
20/09	Concluir as tarefas da Iteração 06, devido a falta de detalhamento dos testes de aceitação da US07 definidos pelo cliente	Alta	Helton	**Vigente**	Contactar o cliente urgentemente
15/10	Redução do período das iterações 05 e 06 devido a greve	Média	Todos	**Vigente**	Rever escopo do planejado e conversar com o cliente a respeito.
15/09	Inserir a tecnologia Hibernate (desconhecida) ao sistema	Média	Todos	**Abortado**	Devido a greve, e a redução de escopo, essa atividades foi abortada.

Observações:

- Data da identificação do Risco;
- Prioridade: Alta - Média - Baixa;
- Status: Vigente - Superado - Abortado.

12.14. Teste de Usabilidade: Roteiro das Atividades

Descrição Inicial

Neste teste de usabilidade, você assumirá as funções de um agente da secretaria de saúde do município de Campina Grande, e atuará como usuário do **PSF*component***. Este sistema possibilita automatizar todo o processo de documentaçao necessário para o bom funcionamento do Programa de Saúde na Família (PSF). O PSF*component* é composto por dois módulos, um deles é responsável pela manipulação das informações do PSF em uma área específica (famílias favorecida, região contempladas, equipe de profissionais responsáveis...), o outro módulo faz controle de medicamentos disponíveis para o PSF (origem e destino dos medicamentos). Como agente de saúde você terá acesso às funções dos dois módulos.

OBS.: Sinta-se à vontade para consultar a ajuda *on-line* ou *off-line* a qualquer instante; Caso encontre alguma dificuldade que não comprometa a realização da atividade, não se preocupe e siga em frente.

Atividade 01: Efetuar *login* no sistema

Roteiro: Nesta atividade você irá entrar no sistema PSF*component*, através de *login* e senha.

Instruções:

- Abra o *browser* de sua preferência;
- Carregue a página do PSF*component* (www.psfcomponent.com.br);
- Informe o seu *login*, sua senha e o município ao qual pertence;
- Acesse o sistema;
- Verifique se está na página principal do PSF*component*

Atividade 02: Cadastrar novas famílias

Roteiro: Nesta atividade você irá cadastrar as informações referentes a duas novas famílias que passarão a ser favorecidas com os serviços do PSF em uma região específica.

Instruções:

- Cadastre a família de Dona Maria preenchendo as informações requeridas no formulário correspondente;
  - Dona Maria do Carmo Silva (35 anos) tem três filhos (Paulo Silva e Sousa (18 anos), Joana Cristina Silva e Sousa (17 anos) e Roberto Carlos Silva e Sousa (10 anos)) e dois netos (Adriana Sousa Rocha (9 meses) e Uilma Sousa Pedrosa (1 ano e meio)). Todos moram na Rua Rodriques Alves, nº 569, Prata.
- Verifique se os dados foram cadastrados corretamente;
- Cadastre a família de Seu Pedro preenchendo as informações requeridas no formulário correspondente;
  - Seu Pedro Targino (55 anos) é casado com Dona Carmélia Targino (50 anos), eles têm 1 filho (João Carlos Targino (26 anos)) e 2 netos (Carla Janaina Targino (9 anos) e Diego Targino (6 anos)). Todos moram na Bela Vista.
- Verifique se os dados foram cadastrados corretamente;
- Volte para a tela principal do PSF*component.*

Atividade 03: Localizar e Editar os dados de um membro de uma família

Roteiro: Nesta atividade você irá buscar por uma pessoa específica (previamente cadastrada nos sistema) e irá editar os dados referentes ao endereço dessa pessoa.

Instruções:

- Procure pelo registro de João Pedro, um morador do Conjunto dos Professores;
- Identifique e selecione o registro requerido;
- Altere os dados referentes ao endereço de João Pedro
  - Rua: <Souto Maior>;
  - Nº: <368>;
- Verifique se as alterações foram efetuadas corretamente;
- Volte para a tela principal do PSF*component.*

Atividade 04: Localizar destino de um conjunto de medicamentos

Roteiro: Nesta atividade você irá localizar o destino de um conjuto de medicamentos que saíram da secretaria de saúde no mês de julho de 2005.

Instruções:

- Dentre os medicamentos listados localize e selecione: Analgésico, Amitryl, Citoneurim, Utracet, ASS e Mercúrio;
- Verifique o destino final de cada um desses medicamentos, sabe-se que todos saíram da secretaria de saúde em Julho de 2005;
- Volte para a tela principal do PSF*component.*

Atividade 05: Localizar qual a equipe de profissionais do PSF, de uma área específica, que atuou em um determinado período.

Roteiro: Nesta atividade você irá identificar quais foram os profissionais da área de enfermagem que estavam de plantão no dia 22 de Maio de 2005 durante o turno da tarde.

Instruções:

- Localize o nome e o telefone de todos os enfermeiros que trabalharam dia 22 de Maio de 2005 das 14 às 19h;
- Ordene o resultado da busca por tempo de serviço;
- Verifique ser o resultado aparece na seguinte ordem:
    - Ana Clara Barbosa - 3333-3333
    - Pablo Pereira da Costa - 3333-2222
    - Cintía Albuquerque - 3333-1111
    - Volte para a tela principal do PSF*component.*

13. Ferramentas Livres de Apoio ao YP

Gerência

Xplanner Ferramenta utilizada para gerência de projetos.

http://www.xplanner.org/

Desenvolvimento

Ant Ferramenta utilizada para a automação de tarefas, tais como compilação de programas ou geração de javadocs.

http://ant.apache.org/

Eclipse Ambiente integrado de desenvolvimento em Java, com suporte a CVS e Ant.

http://www.eclipse.org/

Java Linguagem de programação orientada a objetos desenvolvida pela Sun, com extensões para aplicações cliente/servidor e para dispositivos móveis.

http://java.sun.com/

Testes

Junit Framework para implementação de **testes de unidade** para programas escritos em Java.

http://www.junit.org/

Cactus Framework para **testes de aplicações web**.

http://jakarta.apache.org/cactus

easyaccept Ferrementa para criação e execução de **testes de aceitação**.

http://easyaccept.org/

Integração Contínua

CruiseControl	http://cruisecontrol.sourceforge.net/
Gump	http://jakarta.apache.org/gump/
AntHill OS	http://www.urbancode.com/projects/anthill/default.jsp

Ferramentas com o mesmo objetivo de construção automática.

Controle de Versões

CVS Gerencia alterações concorrentes sobre os arquivos de um projeto.

http://www.cvshome.org/

WinCVS Cliente CVS com interface gráfica.

http://www.wincvs.org/shots.html

Usabilidade

CTTE Ferramenta utilizada para construção do modelo da tarefa segundo o formalismo CTT.

http://giove.cnuce.cnr.it/ctte.html

TERESA Ferramenta para construção de protótipos a partir do modelo da tarefa construído no CTTE.

http://giove.cnuce.cnr.it/teresa.html

iTAOS Ferramenta utilizada para construção do modelo da tarefa segundo o formalismo TAOS.

http://www.dsc.ufcg.edu.br/~itaos/

DENIM Ferramenta para construção de protótipos de interfaces web**.**

http://dub.washington.edu/denim/

DEMAIS Ferramenta para construção de protótipos de interfaces multimídia**.**

http://orchid.cs.uiuc.edu/projects/demais/index.html

Webquest Ferramenta para auxiliar na realização dos testes de usabilidade**.**

http://www.lihm.paqtc.org.br/webquest/

Referências

[1] : XP

http://www.extremeprogramming.org

[2] : RUP

http://pt.wikipedia.org/wiki/RUP

[3] : *Agile Modeling*

http://www.agilemodeling.com/

[4] : DSC

http://www.dsc.ufcg.edu.br

[5] : UFCG

http://www.ufcg.edu.br

[6] : PET

http://www.dsc.ufcg.edu.br/~pet

[7] : LES

http://www.dsc.ufcg.edu.br/~garcia

[8]: *Test-Driven Programming*

http://www.dsc.ufcg.edu.br/~jacques/palestras/tdp/index.html

[9] : Padrões de Projeto

E. Gamma, R. Helm, R. Johnson y J. Vlissides, "*Design Patterns Elements of Reusable Object-Oriented Software*", Addison-Wesley, 1995.

[10] : Padrões de Projeto

http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/map2.htm

[11] : Padrões de Refatoramento

http://www.refactoring.com/catalog/index.html

[12] : *Pair Programming*

http://www.pairprogramming.com/

[13] : iTAOS

CORDEIRO, P. B., *Projeto e implementação do módulo TAME da ferramenta iTAOS para análise e modelagem da tarefa*, Dissertação de Mestrado - COPIN, UFCG, Campina Grande PB, Fevereiro de 2003.

MEDEIROS, F. P. A., *Projeto e implementação do módulo TAOS-Graph da ferramenta iTAOS para análise e modelagem da tarefa*, Dissertação de Mestrado - COPIN, UFCG, Campina Grande PB, Fevereiro de 2003.